COMMODORE MAGAZINE

VOL 2
ISSUE 5

★ COMMODORE ANNOUNCES NEW PRODUCTS

★ COMMODORE'S RANGE OF DISK DRIVES

REGISTERED FOR POSTING AS A PUBLICATION: CATEGORY B

The objective of this magazine is to disseminate information to all users of Commodore computer products. This magazine contains a variety of information collected from other Commodore publications, and generated locally. Contributions from all Commodore User Groups, and individual users are encouraged.

Advertising space is restricted and allocated on a first-come, first-served basis. Advertising rates can be supplied on request.

All copy and advertising should be addressed to:

The Editor,
COMMODORE MAGAZINE,
P.O. BOX 336, ARTARMON
N.S.W. 2064
AUSTRALIA

ISSUE No.	COPY/ADV DEADLINE	PUBLICATION DATE
1	February 18th	March 6th
2	April 1st	April 17th
3	May 13th	May 29th
4	June 17th	July 3rd
5	July 29th	August 14th
6	September 9th	September 25th
7	October 14th	October 30th
8	November 25th	December 4th

Production & Typesetting: Mervyn Beamish Graphics Pty. Ltd.
82 Alexander Street, Crows Nest
2065
Phone 439 1827

Printer: Liberty Print
108 Chandos Street, Crows Nest
2067
Phone 43 4398

SUBSCRIPTIONS

	Annual Subscription	Single Copy
Postage paid within Australia	$A30.00	$A5.00
Overseas Postage Paid	$A38.00	$A6.00

Subscriptions to COMMODORE MAGAZINE can be obtained from your authorised Commodore Dealer, or from:

COMMODORE MAGAZINE,
P.O. BOX 336, ARTARMON,
N.S.W. 2064,
AUSTRALIA.

Vol 1 1981
Vol 2 1982

Typeset and assembled off Commodore Wordcraft disks

PLEASE NOTE: To provide a good information service to Commodore users, we will regularly mention equipment, software and services offered by companies and individuals not directly related to Commodore. In doing so, we are not making recommendations, and cannot be responsible for the validity and accuracy of any statements made.

EDITOR'S DESK . . .

There is a lot of excitment within Commodore these days with so many new products available for the Commodore range of computers.

It is encouraging to see so many peripherals and software packages available and these will be reviewed in the Commodore Magazine after our evaluation.

This issue brings the first news of the Commodore 64 which was shown at DATA 82 and the NCC in Hobart during August. The initial reaction has been tremendous, with software suppliers showing a keen interest in developing or transferring programs onto the Commodore 64. The sprite graphics allow new dimensions in personal computing and combined with a music synthesizer, this new product will attract many more Commodore users.

Please continue to send in your letters for information on any subject you would like covered in this magazine. With a mixture of PET/CBM and VIC readers there are many subjects that must still be clouded in mystery. Drop us a line and we will help to solve your Commodore problems.

table of contents

Page *Contents*

2 Commodore News
3 Letters to the Editor
4 COMMODORE 64
5 COMMODORE MAX Machine
6 Comparison of Features of 64 & MAX
7 Using the VIC-20 for Forms
9 Programming Tips
11 A Userable RESTORE
13 VIC Interface & Programming Cartridges
15 Cartridge Development
16 Using the Programmable Character Set Editor
17 Fast Forward on the VIC Cassette
18 Commodore's Range of Disk Drives
20 Using the VIC Expansion Module
21 4022 Printer Notes
22 Identifying the Disk Drive
23 Software Controlled Immediate Mode
26 Machine Language Auto Locate
29 Disk Utilities
33 VIC Memory Allocation Tables

next issue:

VIC/PET/CBM Keyboard Differences
Using Printer Formatting
Commodore Hard Disk Drives

list of Advertisers

B.S Microcomp	inside Cover
Compute CBM Systems	Back Cover
C.W. Electronics	Page 12
Mervyn Beamish Graphics	Page 19
Micropro Design	Page 19
The Microcomputer House	Page 8

COMMODORE NEWS

NEW STORAGE PERIPHERALS ANNOUNCED

Increased storage capacities are now possible with the latest addition to the Commodore range of peripherals.

The CBM 8250 gives double sided floppy diskette storage totalling 2 Megabytes. A new version of the DOS allows relative file capacity equal to the storage capacity of one diskette.

The CBM 9060 and 9090 are both hard disk drives using the latest Winchester technology to give 5 and 7.5 Megabytes of storage. They are in a compact housing and utilise the same disk structure as the Commodore floppy disk drives to give software compatability.

Most of the Commodore software is being converted to support the hard disk units and with the additional storage capacities, powerful business systems can be configured at economical prices.

STOLEN

Help!! A Commodore 8032 was stolen from a house in Cranbrook, a suburb of Sydney, during August. Richard had just arrived from the U.K. with his Commodore system, to be greeted by a burglar with a fancy for the CPU only.

The serial number of the 8032 is WG 024575 and is distinguishable by the fully optioned inside; high res board, PET Clock, EPROM expansion, and Commando chip. Richard is very keen to be reunited with his 8032 computer and is offering a recovery fee of 10% of the value.

If you see Richard's computer, please contact him by phoning (02)669-0411 extension 336 and asking for Richard Greenaway.

NEW ARCADE-STYLE GAMES FOR POPULAR COMMODORE VIC-20

Exciting new cartridge games, including three Bally/Midway arcade games, for the phenomenally popular VIC-20 home computer have been announced by Commodore.

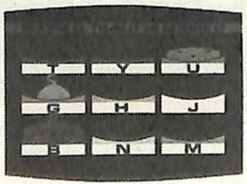
The Bally/Midway games on cartridge include Gorf, Omega Race and Wizard of Wor and are real arcade games. High resolution graphics, sound effects and play action are exactly like the popular Bally/Midway coin-operated games. Commodore's own Space Vulture, Raid on Fort Knox and Pinball Spectacular, also planned for release in the near future, have the same kind of fast action and exciting graphics as the arcade games. Mole Attack and The Sky is Falling, Commodore's new childrens' cartridge games, engage tots with faster and faster action.

Since the introduction of the VIC-20, its capabilities have continued to grow at a tremendous rate. VIC users can now learn the fundamentals of computer programming at home using Commodore's "Introduction to BASIC" manuals and tapes. More advanced programmers have great flexibility for designing programs and creating graphics and sound effects, with the use of the Super Expander cartridge, Programmable Character Set Editor and VICMON machine language cartridge.

GORF

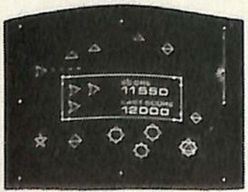
SPIDERS OF MARS

JUPITER LANDER

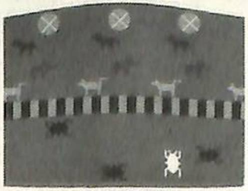


MOLE ATTACK

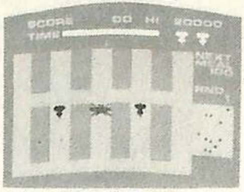
VISIBLE SOLAR SYSTEM



OMEGA RACE

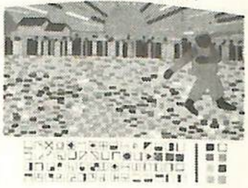


MENAGERIE



RADAR RAT RACE

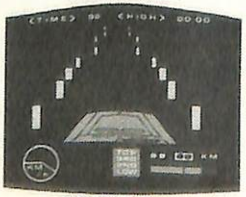
HOME BABYSITTER



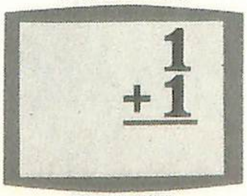
LIGHTPEN ARTIST

WIZARD OF WOR

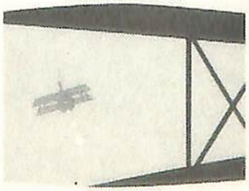
AVENGER



MIDNIGHT DRIVE

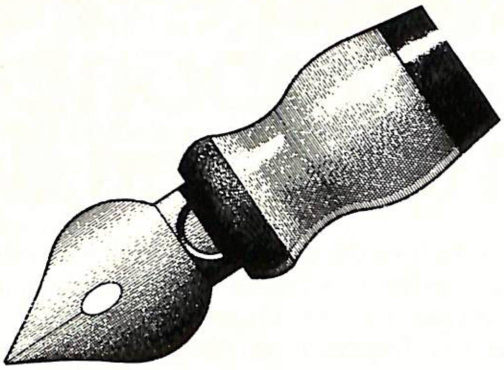


MATH TEACHER



ACE OF ACES

LETTERS TO THE EDITOR



Dear Editor,

As a VIC owner there are features that other users may not be familiar with.

VIC owners should be well aware that by holding down the shift key and the Commodore key (the little flag), the character set changes. One set gives upper case (capitals) and when shifted, graphics characters. The other gives upper and lower case characters. Other graphics characters are available at all times by pressing the Commodore logo key.

However there may be times when you would not want this swapping to occur. For instance, you may have a user defined character set.

There is a way to 'lock out' this swapping. When the VIC detects the Shift-Commodore combination, it first checks memory location 657. If the content is 127 or less, the change occurs. If the content is greater than 128, then the change will not happen.

So to disable swapping POKE 657,255

To enable swapping POKE 657,0

There is another way using control codes.

To disable PRINT CHR$(8)

To enable PRINT CHR$(9)

Regards,
Brad Fisher.

Dear Editor,

I have just bought a copy of 'WORDCRAFT'. I found some major flaws in the software manual and I wonder if one of your readers can help me understand it.

These are some of my probleems;

1. I can't understand the words in the manual.
2. The index is of little help.
3. The damn thing is awkward to use.

Let's take each point in turn. Firstly, here is a sentence from page 54;

"The text-associated controls are normally placed above the text character position in which they occur, e.g., the t symbol in the above example appears in character position 23 which is where it was used."

Now I simply don't understand that at all. Can someone help?

Second, the index; I know that it is possible to put together a mailing list on 'WORDCRAFT'. This is nowhere to be found in the index. And there are many other ommisions.

Thirdly, trying to balance a large plastic folder on my knee and type at the same time is a little difficult.

So I would appreciate it if someone has simplified the procedure and is willing to share.

Yours faithfully,
Mike Connolly,
PO Box 458
ARTARMON 2064.

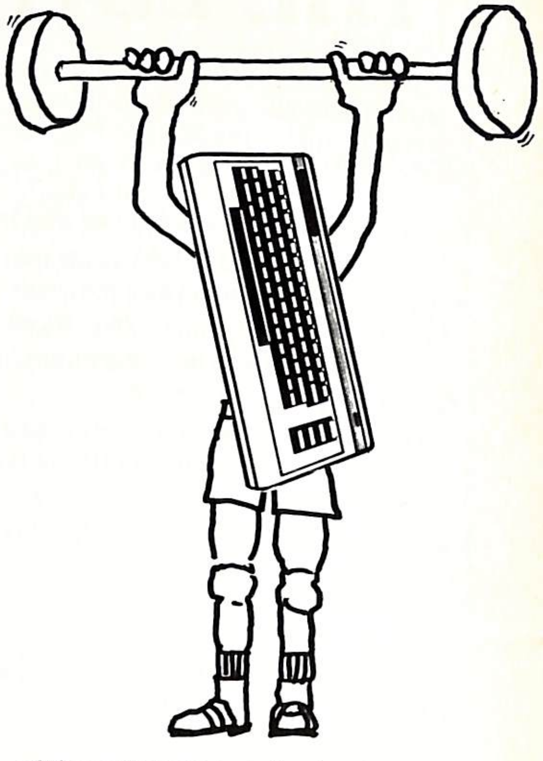
It seems you have not had previous experience in wordprocessing and that no training has been given when you purchased the WORDCRAFT package.

Wordprocessing has its own vocabulary, very similar to the microcomputer, and unless you are familiar with many of the words used, it is difficult to get straight into a package as powerful as WORDCRAFT.

Continued on Page 14

Powerful – Versatile COMMODORE-64 Computer Ideal for Home and Business Applications

The Commodore-64, one of Commodore's newest breakthroughs in adding versatility and power to home and business microcomputing, was unveiled at the Summer Consumer Electronics Show in Chicago and The National Computer Conference in Houston.



The Commodore-64, which Shearson/American Express said "could be the microcomputer industry's outstanding new product introduction since the birth of the industry, "gives home and business users powerful 64K computing capability along with exciting three-dimensional-style colour game graphics and music synthesis. The suggested retail price will be under $1000.00 offering the Commodore-64 one-third more computing power than the Apple II+® at a much reduced price.

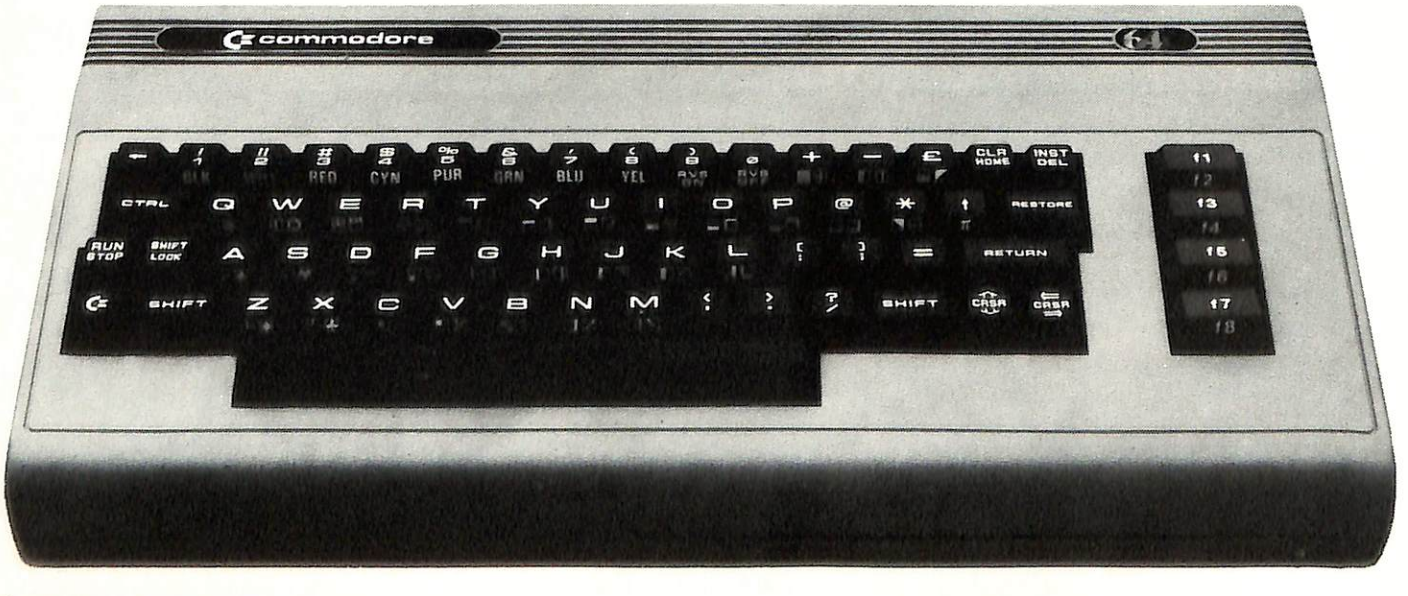
Physically resembling Commodore's popular VIC 20 computer, the Commodore-64 can use the VIC 20 peripherals, and goes far beyond the VIC's capabilities. With the addition of a cassette interface, the Commodore-64 can use many programs and files created for Commodore's sophistocated line of PET and CBM education – and business – oriented computers. And with the addition of an IEEE cartridge, the Commodore-64 can run any Commodore peripheral, including a dual disk drive and a CBM printer.

For those programs that need conversion, the Commodore-64 has a PET emulator that makes it operate like a PET in most areas. One of its most impressive features is its ability to use a Z80 cartridge to run CP/M® giving users access to one of the largest collections of microcomputer software available.

Computerists can use the Commodore-64's versatility to synthesize music and play all the exciting games designed for the new Commodore MAX Machine. Its 40 column x 25 line screen and 16 colours give it a flexibility for users to create their own graphics characters and games. The sprite graphics allow many different shapes to be moved around the screen either in front or behind other sprite shapes.

With 64K Random Access memory (RAM) the Commodore-64 has the capability for sophisticated home and business applications. Word processing programs will be available for the Commodore-64, along with Commodore's own new "electronic spreadsheet". Many new applications will be on the market soon.

The Commodore-64 relies on state-of-the-art integrated circuits designed and produced by Commodore's subsidiary, MOS Technology. The heart of the Commodore-64 system is a new microprocessor, the 6510. The 6510 is similar to the MOS 6502 – the chip that made microcomputing a household occupation – but contains additional input/output (I/O) lines to handle the processing required by the new system.



COMMODORE MAX MACHINE,

3 in 1 Computer-Game Machine-Music Synthesizer.
Sure to Change the Old-Fashioned Video-Game Buying Habits of Consumers.

The Commodore MAX Machine, a revolutionary three-in-one home computer-game machine-music synthesizer, was recently announced by Commodore Business Machines in the United States.

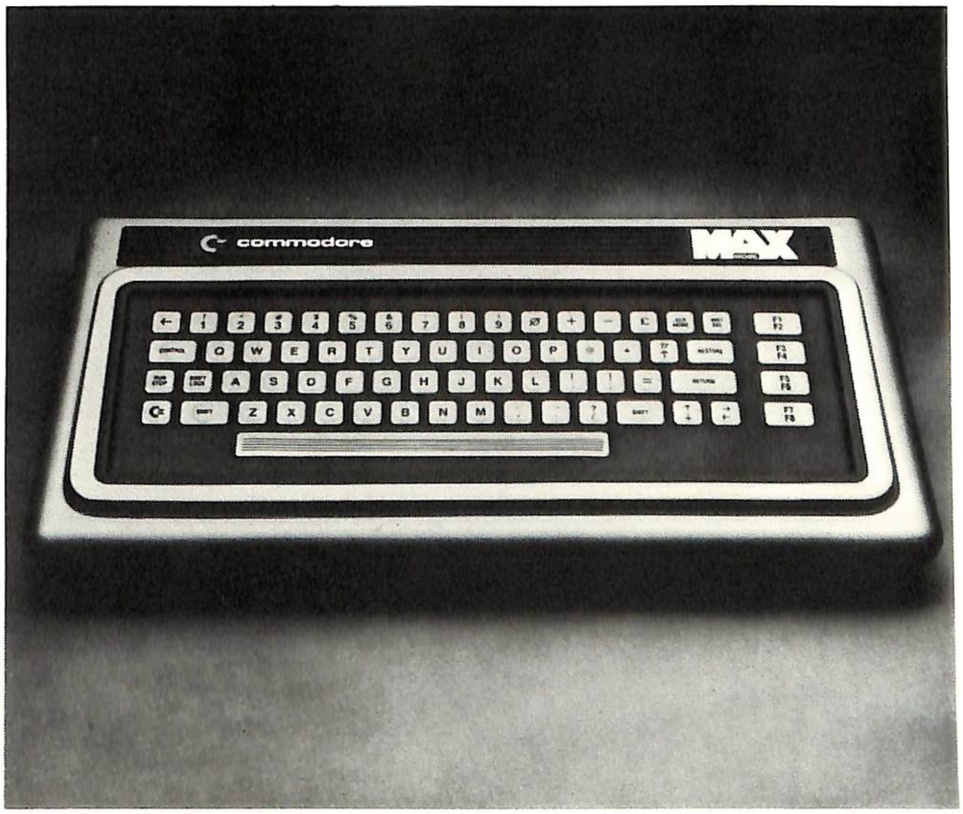
The MAX Machine, shown in prototype at the Winter Consumer Electronics Show under the code name ULTIMAX, drew raves for its extraordinary versatility, price/performance ratio and three-dimensional-style colour game graphics. Now encased in an innovative futuristic housing, it will be sold next year with an array of arcade games, as well educational and musical programs. The suggested retail price will be about the same as an ordinary television game machine – the Commodore MAX Machine is certain to change consumer's old-fashioned video-game buying habits.

Relying on a new display chip designed by Commodore's subsidiary, MOS Technology, the MAX Machine produces colour and graphics formerly available only with a highly sophisticated character generator. And because the MAX is a real computer, users do not have to rely on pre-programmed games, but can actually create their own games, then save them on cassette tape for future use. Its 40 column x 25 line screen and 16 colours give the MAX great flexibility for unique and exciting graphics.

Using a new Sound Interface Device, the Commodore MAX Machine produces music and sound effects that rival many of the best music synthesizers now available. The MAX produces three independent voices, each with a nine-octave range, contains a programmable ADSR (attack, decay, sustain, release) generator and a programmable filter, and has variable resonance. With these sophistocated features, the MAX Machine can command astounding orchestration when it is used with a good quality audio system.

With a BASIC language cartridge, Commodore MAX Machine users can learn the fundamental language of computing and write their own programs using simple BASIC commands. MAX Machine BASIC can be translated for use with all other Commodore systems, and is capable of handling everything from word strings to math functions. With the MAX Machine's nine-digit numeric accuracy and built-in math functions, users can write a variety of useful programs for home applcations.

The heart of the MAX Machine system is a new microprocessor, the 6510, designed by Commodore's MOS subsidiary. There are many additional features compared to the 6502 chip, also designed by MOS, such as additional input/output (I/O) lines to handle the processing required by the new system.



COMMODORE Quick Reference Fact Sheet

	MAX Machine	Commodore-64
Memory	2K built-in plus memory Cartridge as per BASIC specifications	64K built-in 16K ROM add-on
Screen Size	40 col x 25 lines	40 col x 25 lines
Graphics	320 x 200 pixels 255 foreground/ background color combo 16 text colors 8 movable Sprites	320 x 200 pixels 255 foreground/ background color combo 16 text colors 64 graphic characters 256 movable Sprites Independent magnification Extended Hi-res modes
Sound	6581 Sound Interface Device (SID) 3 independent voices, 9 octaves each Programmable ADSR Programmable filter Variable resonance & master vol. control	6581 Sound Interface Device (SID) 3 independent voices, 9 octaves each Programmable ADSR Programmable filter Variable resonance & master vol. control
Peripherals	Datassette Joystick Double paddle Lightpen	Datassette Joystick Double paddle Lightpen Z80 microprocessor on cartridge VIC MODEM VIC disk drive VIC graphic printer With IEEE-488 cartridge: All Commodore peripherals
Compatability	Same game cartridges will work on Commodore-64	Other BASIC Commodore programs easily converted. PET emulator to be available.

Using Your VIC 20 to Design Useful Forms

The VIC 20 is a great game machine, but games are not its only application. You can use your VIC and its powerful graphic capabilities to design many types of forms, for both business and home. The form below, which allows the user to keep track of incoming and outgoing inventory, was printed on the VIC 1515 Graphic Printer. The blocks were created by piecing together graphic characters straight from the VIC's keyboard.

Let's talk more about modifying the program to suit your individual needs. Line 5 is probably the only line you **won't** want to change. This line directs the output to the printer. Lines 10 to 50 are the "personal" touches to the program. Line 10 prints the title of the form in enhanced letters. CHR$ (14) directs the printer to print in large boldface letters, and CHR$ (15) returns the printing to normal size. Line 18 simply prints a place to enter the date. Lines 30 and 40 print column titles. All of these names and titles can be easily changed to design your own forms.

The second half of the program prints the graphic blocks. The accompanying chart describes how to obtain the special characters from the VIC keyboard.

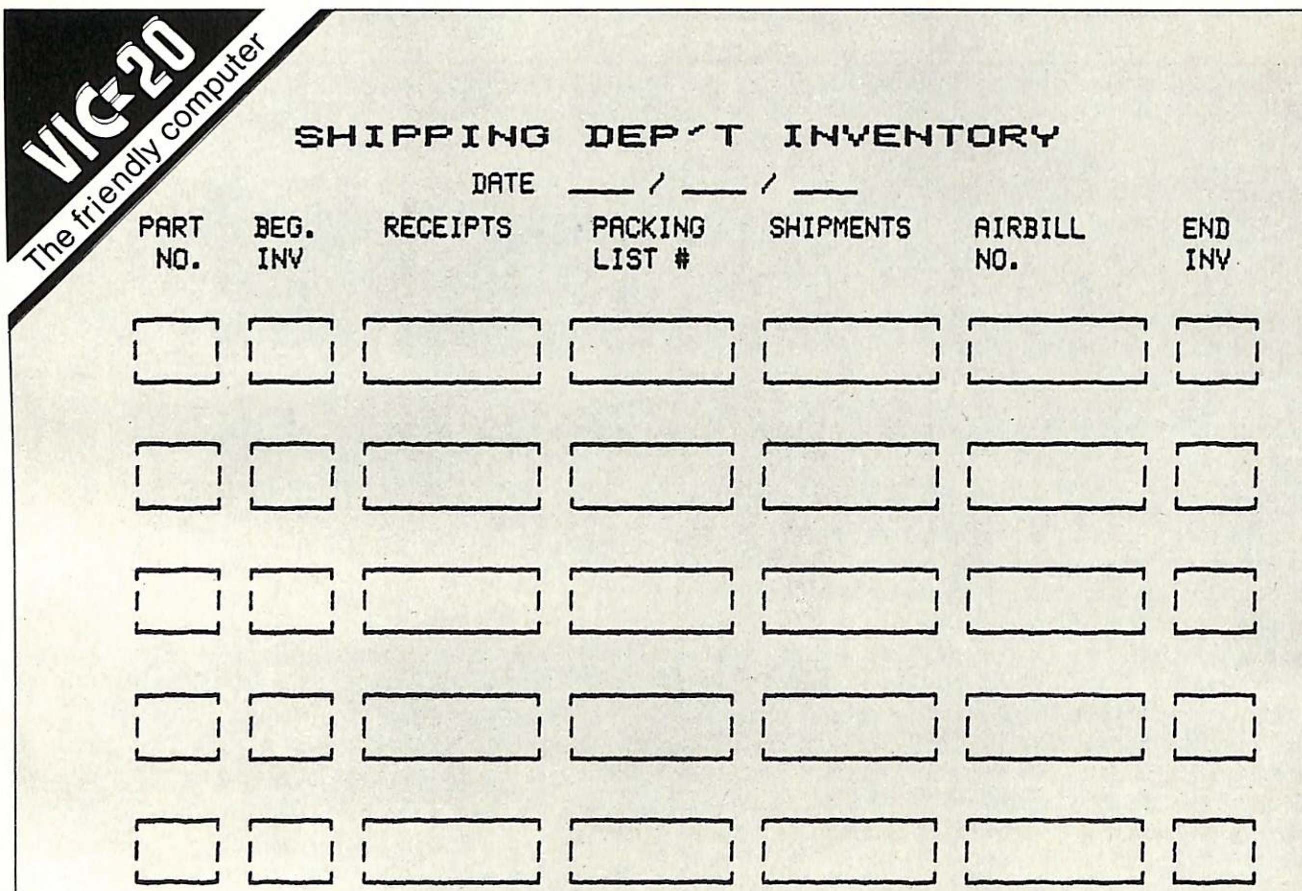
GRAPHIC CHARACTER	**KEYS**
┌	C= A
─	SHIFT *
┐	C= S
└	C= Z
┘	C= X
│	SHIFT —

These characters are used in lines 150 to 170. The size of the blocks can be changed by using different combinations of symbols. Changing the loop variable in line 100 will allow you to have more rows of blocks.

This program, with minor modifications, will allow you to keep track of just about anything, from your slide collection at home to your incoming packages at work. Next month we'll feature an enhanced version of this program that will allow you to input your data and have your VIC fill in the blocks automatically.

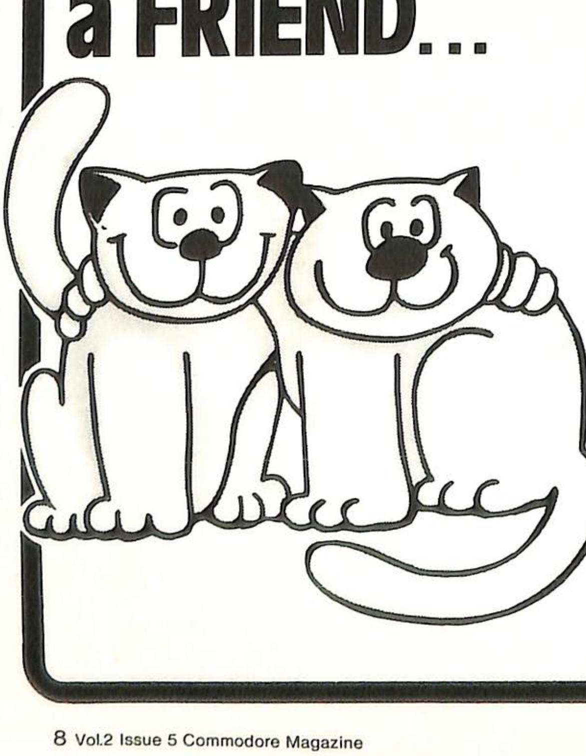
—Joseph Siciliano

```
2 REM ** VIC BUSINESS FORMS **
3 REM ** BY JOSEPH SICILIANO **
5 OPEN4,4:CMD4
10 PRINTCHR$(14)"      SHIPPING DEP'T INVENTORY"CHR$(15)
15 PRINT:PRINT
18 PRINT"                       DATE  ___ / ___ / ___"
20 PRINT:PRINT
30 PRINT"   PART   BEG.    RECEIPTS     PACKING    SHIPMENTS    AIRBILL      END
"
40 PRINT"    NO.   INV                  LIST #                  NO.          INV
"
50 PRINT
100 FOR A=1TO5
150 PRINT" ┌───┐ ┌───┐ ┌─────────┐ ┌────────┐ ┌─────────┐ ┌─────────┐ ┌──
──┐"
160 PRINT" │   │ │   │ │         │ │        │ │         │ │         │ │
 │"
170 PRINT" └───┘ └───┘ └─────────┘ └────────┘ └─────────┘ └─────────┘ └──
──┘"
180 PRINT
200 NEXT A

READY.
```



```
        SHIPPING DEP'T INVENTORY
                  DATE ___ / ___ / ___
PART  BEG.   RECEIPTS   PACKING   SHIPMENTS   AIRBILL   END
 NO.  INV               LIST #                NO.       INV
READY.
```

Every PET needs a FRIEND...



Introducing a new series of programs that are 'FRIENDLY' to the user and represent outstanding value for money. . . .
and there will be more 'FRIENDS' coming

FRIEND 1 - Word Processor

An on-line ROM chip for 8, 16, and 32K machines. This Word Processor program has been written by professionals specially for The Microcomputer House.
The program can be used with or without a printer and will be available shortly in disk and tape versions as well.

WP CHIP	$85
DISK	$70
TAPE	$60

FRIEND 2 - Mailing List

This is a dual disk based system for the 4000 series microcomputers. It caters for 2,100 records per data disk and offers sort and select facilities. It will also be available shortly for single disk systems.
MAILING LIST $85

FRIEND 3 - Data Handler

This is a ROM chip containing a machine language routine which allows the programmer to control screen input. ● Alpha Field Entry ● Numeric Field Entry ● Date Entry ● Disk Fastget ● Field Reverse ● Field Flash. Each of these functions have different options. FRIEND 3 is a derivative of our security ROM used by all our packages.
4000 series and 8000 series $85 each.

All programs come with a complete instruction manual.

DEMONSTRATION STOCK AVAILABLE AT NEVER TO BE REPEATED PRICES
JUST PHONE OR CALL IN

Bankcard
Mail Orders
Welcome

The Microcomputer
MTH House Pty. Ltd.
1ST FLOOR, 133 REGENT STREET
CHIPPENDALE N.S.W. 2008 PHONE (02) 699 6769

PROGRAMMING TIPS

by Paul Higginbottom

First I would like to answer some frequently raised questions about screen formatting of data, and then take a look at a few techniques to make programs smaller and more elegant.

Number Juggling.

Formatting numbers on the screen can cause problems when the TAB function is used. If numbers are to be printed in columns, then it would be nice to ensure that the decimal point of the numbers always line up in the column.

For example, if the number is simply TABbed onto the screen, and the number is a "0", then it will appear at the left hand side of the column, which doesn't look very smart. It is therefore necessary to use the length of the number (ie. the number of digits including decimal points) to drive the TAB expression. A number has a leading space and a trailing cursor-right which needs to be taken into consideration in a string. In order to use LEN it is first necessary to convert the number into a string using STR$. The number of characters in the number is given by:

```
X = LEN (STR$ (A) ) - 1
```

...where A is the number. The trailing cursor right is ignored by STR$ but 1 is subtracted to take account of the leading space. So now, taking the above example, if we were to TAB (10-X) we would be in business, right? No, not quite yet.

There are a couple more possibilities that should be considered. Sometimes it is desirable to tack leading zeroes onto integer numbers (ie. to display "0038" rather than "38"). With decimal numbers, we may want a specified number of digits following the decimal place, followed by trailing zeroes which the PET does not do for us. All of this will affect the positioning of numbers output to the screen.

Without question, the easiest way to handle number formatting is to first turn the number into a string:

```
A$ = MID$ ( STR$ (A), 2)
```

STR$ converts "A" to a string and the MID$ function is used to take the 2nd character onwards, thus removing the leading space. To add leading zeroes we use the RIGHT$ function:

```
A$ = RIGHT$ ("000000" + A$, 4 )
```

Assuming "A" is an integer, the above will produce a 4 character string. The number of leading zeroes will be dependent on the size of "A" (ie. "16" produces "0016" and "1024" stays "1024"). For compactness, all of the above could have been done with:

```
A$ = RIGHT$ ("000000" + MID$ ( STR$(A),2, 4)
```

Rounding decimal numbers and adding trailing zeroes is a little trickier. First we must decide how many significant digits are to follow the decimal place. The following is an example for 2 significant digit truncating:

```
A = 1035.55534
A = INT (A * 100) / 100
PRINT A
1035.55
```

The above merely moves the number two places to the left, chops off the fractional part, and then moves it back two places to the right. However, this is not rounding but rather truncating, which is not the same. For rounding, we must first decide what degree of rounding is desired. Most often, numbers are rounded to 2 decimal places, or "to the cent". Our example then becomes, simply:

```
A = 1035.55534
A = INT (A * 100 + .5) / 100
PRINT A
1035.56
```

The same result would be accomplished by first adding .005 to A and multiplying that by 100, but BASIC is more accurate at decimal arithmetic that lies closer to the decimal point.

Numbers that end up with 1 decimal or less will need trailing zeroes. Once again, this can be done with a string manipulation. For example:

```
A = 1035.59534
A = INT ( ABS(A) * 100 + .5) / 100
PRINT A
1035.6
S$ = CHR$ (32 - (V/t0) * 13)
A$ = MID$ ( STR$ (INT(A)) ,2)
DP = INT ((A - INT(A)) * 100 + .5)
A$ = A$ + "." + RIGHT$ ("00" + MID$
                            (STR$(DP) ,2) ,2)
A$ = RIGHT$ (" " + S$ +A$ ,10)
PRINT A$
1035.60
```

This probably looks sort of clumsy but it's designed to do everything; trailing zeroes; pos. and neg. rounding; and decimal point alligning. Notice that we've taken the ABSolute value of A before entering the routine. S$ will be either a space or a minus sign. Then we grab the INTeger part of A into A$. DP is used to take the decimal part of A, and round it on the left of the decimal point. Next we build it all together by taking the integer part, adding our own decimal point, followed by the decimal part. As usual, the leading space is stripped off DP with MID$. "00" is added to this (remember, DP could have a value of say 04 or an even 0) and RIGHT$ comma 2 gives us our 2 decimal places. Lastly we add some leading spaces (which could also be zeroes) and then we stick the sign S$ on the front.

Easy, right? Well, you might have done it differently but this was a string juggling exercise for the practice.

Variable Flip-Flop.

Programs can be shortened a great deal with a little thought and an active imagination. For example, it is often necessary to set a flag if a condition is met or to compliment the flag. One might code:

```
1200 IF FLAG=0 THEN FLAG=1 : GOTO 1220
1210 FLAGO
1220 ...
```

On consideration, the statement:

```
1200 FLAG = 1 – FLAG
1210 ...
```

will be seen to have the same effect.

Screen Codes to ASCII.

This program is a screen dump routine which makes it possible to copy the contents of the screen onto the printer. This subroutine has been presented several times but we'll be looking at technique as opposed to operation.

```
5000 OPEN 4, 4
5010 FOR J = 0 TO 999
5020 P = PEEK (32768 + J)
5030 GOSUB 5500
5040 IF P< 64 THEN PP+ 64 : GOTO 5090
5050 IF P<126 THEN P=P+128 : GOTO 5090
5060 IF P/t128 THEN PP+ 64 : GOTO 5090
5070 IF P/t191 THEN P=P- 64 : GOTO 5090
5080 IF P=255 THEN P= 191 : GOTO 5090
5090 PRINT #4, CHR$(P);
5100 X =X+1 : IF X=40 THEN PRINT#4 : X=0 :
                                       F=0
5110 NEXT
5120 CLOSE 4
5130 RETURN
5500 REM REVERSE FLAG
5510 IF F1 AND P>127 THEN 5550
5520 IF P>127 THEN F=1 : PRINT#4,"[RVS]";
                                : GOTO 5550
5530 IF F=0 AND P<127 THEN 5550
5540 IF P<127 THEN F=0 : PRINT#4,
                          "[RVS OFF]";:GOTO5550
5550 RETURN
```

As you can see, the routine begins PEEKing the screen into P. The subroutine at 5550 deals with reverse field characters. Then P is converted to it's corresponding CHR$ value and is sent to the printer. Lines 5010 and 5100 are set up for 40 columns but this can be easilty changed to 80.

Now let's have a closer look at those five nasty IF statements. If we look at the differences between PEEK/POKE codes and ASCII, it becomes apparent that only the 3 most significant bits (bits 5 [32], 6 [64], and 7 [1281]) are changed. "Aha, a bit of boolean algebra will solve this problem!" Using the OR and AND functions, it is possible to make the conversion with just one line! Thus if P=PEEK(32768), the top left corner of the screen, then:

```
C = (9AND 127) OR ((P AND 64)*2) OR ((64-P
AND 32)*2)
```

Where C is the corresponding ASCII character. Here's the new program:

```
5000 OPEN 4, 4 : T=40 : S=32768
5010 FOR J =S TO 33767 : P=PEEK(J)
5020 R$ = CHR$(146-(P AND 128))
5030 R$=LEFT$(R$,-(P>127 AND
        PEEK(J-1-(J=S))/t128 OR P<128 AND
PEEK(J-1-(J=S))>127))
5040 P=(P AND 127) OR((P AND 64)*2)
    OR((64-P AND 32)*2) -((P AND127)=0*64
5050 PRINT#4, R$;CHR$(P);
5060 IF(J-32807)/T=INT((J-32807)/T) THEN
                                  PRINT#4
5070 NEXT : CLOSE4
```

The extra little bit at the end of 5040 takes care of "@" signs that have a screen code of zero. Lines 5020 and 5030 generate a RVS ON or RVS OFF character and then decides whether to send it or not (5030). This is dependent on the field of the last character. The routine has one bug though; if quotes are printed to the printer, any RVS or RVS OFF characters sent will appear literally. If you expect there may be quotes, you'll need to modify so that a CHR$(141); is sent to do a carriage return with no line feed. Then you'll need to position back to where you left off.

This routine is dreadfully slow but it was meant to be an exercise in boolean algebra. Several screen print utilities have been released that are in machine code.

CHANGING THE DEVICE NUMBER ON DISK UNITS.

by Garry Mason

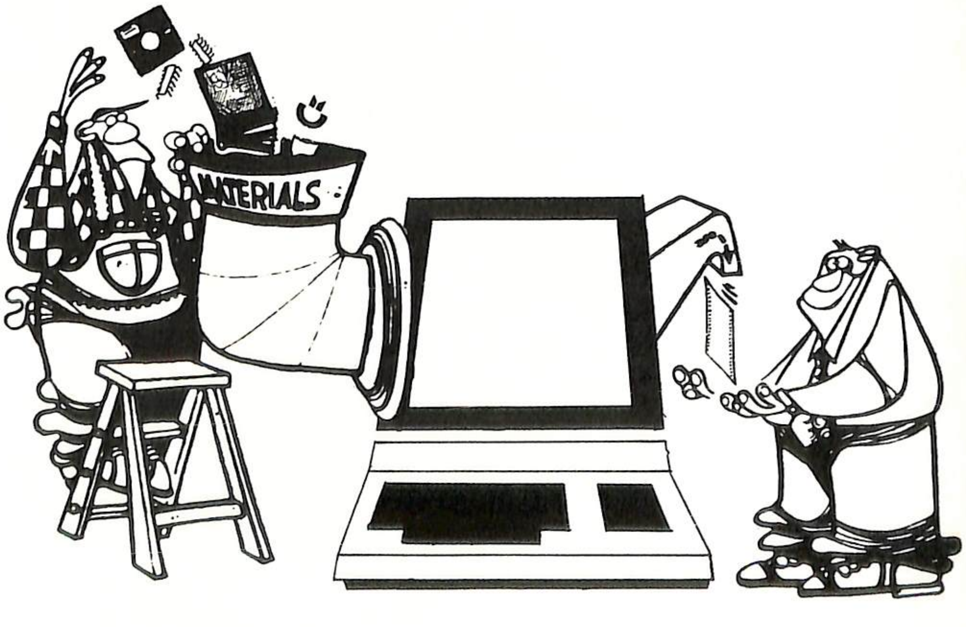
The address on all disk units is normally 'hard wired' to unit 8 on the IEEE bus, but it can be changed with a software command to any number between 8 and 11 on all units except the single disk units which have a range of between 8 and 15.

This allows more than one disk unit to be on line to any one CPU, with the devices being dynamically assigned and readily accessable under the new unit number.

This can be accomplished by the following sequence of instructions. DO is the old device number (normally 8) and DN is the new device number.

```
10 LB= 12 (for 8050, 8250)
   LB= 50 (for 4040, 3040)
   LB= 119 (for 2031, 1540, 1541)
20 DO= 8
30 DN= 9
40 OPEN 15,DO,15
50 PRINT#15,"M-W"CHR$(LB)CHR$(00)
     CHR$(2)CHR$(DN+32)CHR$(DN+64)
60 CLOSE 15
```

The program writes two bytes (dn+32 & dn+64) to the DOS common area of $000C. These variables are called by the system as LSNADR (listen address) and TLKADR (talker address) in the IEEE handshake routines.



A USERABLE RESTORE.

by Brad Fisher

If any of your programs use DATA statements extensively, then you have probably wished at some time or another to be able to say:

```
160 IF A=1 THEN RESTORE 200
```

and have line 200 as your next source of DATA. Unfortunately, Commodore BASIC doesn't allow this. Here are a number of ways to do this (almost) ranging from the simple to the very simple.

Often we want to choose between two or more blocks of DATA. The end of each DATA-block must be flagged with a unique value. e.g. 99999 or XYZZX. If we have done this we can use the following subroutine:-

```
8000 RESTORE
8010 READ A:IFA<>99999 THEN GOTO 8010
8020 RETURN
```

To start from the first DATA-block, GOSUB8000. To bypass the rest of the current DATA-block to reach the start of the next, simply GOSUB8010 'But I knew that!' you say.

If your subroutine is not near the start of your program, or you have a mammoth number of DATA statements, this could be quite time consuming. A slightly quicker way would be to use a FOR-NEXT loop. Since BASIC stores the actual return address of the loop, no time is lost searching for the correct line number (8010 in the above example).
Try this:-

```
8000 RESTORE
8010 FOR XZ=0TO1 STEP 0
8020 READ A: IFA=99999 THEN XZ=1
8030 NEXT:RETURN
```

'Step zero?' you ask, 'Yes' I answer

Because the step value is zero, the FOR loop will continue at infinitum, unless we exit the loop prematurely or change the index ourselves. With this subroutine, the end of the DATA-block is found quite quickly.

The third method is the simplest of all (just a teensy bit of a lie!) BASIC stores the address and the line number of the next available DATA statement in zero page. What could be simpler than to juggle these locations and make BASIC READ what we want it to READ? However, it must be done in machine code (and this method is the fastest). It will be accessed with a SYS call. e.g. SYS 826,[line number].

In the code we will first need to check for a comma, to seperate our parameters. Then we have to evaluate the expression on the right. This means we can say SYS 826,(4*I+k) etc.

Next we have to check whether the line number is in range. BASIC needs to know the address of this line number, so we must search for it (with an error message if there is no such line).

Lastly we place the chosen line number and it's address in the correct place in zero page.

Simple.

Now here's how to do it.

BASIC 2.0

```
JSR $CDF8 /check for comma
JSR $CC8B /evaluate expression
JSR $D6D2 /check valid line number
LDA $12   /place line number in
STA $3D   /'current data line
LDA $11   /number' locations
STA $3C
JSR $C52C /find address
BCC $0D   /not found?
LDA $5D   /convert address
SBC#$01   /to correct value
STA $3E   /and store in
LDA $5D   /current data
SBC#$00   /address' pointer
STA $3F
RTS       /return if ok
LDX#$31   /print error message
JMP $C357 /and exit
```

HEXDUMP

```
.:20 F8 CD 20 8B CC 20 D2
 .:D6 A5 12 85 3D A5 11 85
.:3C 20 2C C5 90 0D A5 5C
 .:E9 01 85 3E A5 5D E9 00
 .:85 3F 60 A2 31 4C 57 C3
```

BASIC 2.0 OBJECT CODE LOADER

```
1    DATA 32,248,205, 32,139,204, 32,210,       1102
2    DATA214,165, 18,133, 61,165, 17,133,        906
3    DATA 60, 32, 44,197,144, 13,165, 92,        747
4    DATA233, 1,133, 62,165, 93,233, 0,          920
5    DATA133, 63, 96,162, 49, 76, 87,195,        861
10    DIMBYTE(7)
100   PRINT"
110   PRINT" THIS ADDRESS USED TO CALL RESTORE"
120   PRINT"
130   SA=VAL(SA$):IFSA<1ORSA>49152THENPRINT
140   FORLINE=1TO5
150   SUM=0
160   FORJ=0TO7:READ BYTE(J)
170   SUM=SUM+BYTE(J):NEXT
180   READ CHECK
190   IFSUM<>CHECKTHENPRINT
200   PRINT"DATA LINE #";LINE;"OK."
210   FORJ=0TO7:POKEA+J,BYTE(J):NEXT
220   SA=SA+8:NEXT
READY.
```

VIC BASIC V2

```
JSR $CEFD /check for comma
JSR $CD8A /evaluate expression
JSR $D7F7 /check valid line number
LDA $15   /place line number in
STA $40   /current data line
LDA $14   /number' locations
STA $3F
JSR $C613 /find address
BCC $OD   /not found?
LDA $5F   /convert address
SBC#$01   /to correct value
STA $41   /and store in
LDA $60   /'current data
SBC#$00   /address' pointer
STA $42
RTS       /return if ok
LDX#$18   /print error message
JMP $C437 /and exit
```

HEXDUMP

```
.:20 FD CE 20 8A
.:CD 20 F7 D7 A5
 .:15 85 40 A5 14
 .:85 3F 20 13 C6
 .:90 0D A5 5F E9
 .:01 85 41 A5 60
 .:E9 00 85 42 60
 .:A2 18 4C 37 C4
```

VIC BASIC V2 OBJECT CODE LOADER

```
1    DATA  32,253,206,  32,138, 661
2    DATA 205,  32,247, 215,165, 864
3    DATA  21,133,  64, 165,  20, 403
4    DATA 133,  63,  32,  19,198, 445
5    DATA 144,  13,165,  95,233, 650
6    DATA    1,133,  65, 165,  96, 460
7    DATA 233,   0,133,  66,  96, 528
8    DATA 162,  24,  76,  55,196, 513
10 DIMBYTE(4)
100 PRINT"
110 PRINT"THIS ADDRESS USED TO":PRINT"CALL RESTORE"
120 PRINT"
130 SA=VAL(SA$):IFSA<1ORSA>49152THEN PRINT"
140 FORLINE=1TO8
150 SUM=0
160 FORJ=0TO4:READ BYTE(J)
170 SUM=SUM+BYTE(J):NEXT
180 READ CHECK
190 IFSUM<>CHECKTHEPRINT"
200 PRINT"DATA LINE #";LINE;"OK."
210 FORJ=0TO4:POKEA+J,BYTE(J):NEXT
220 SA=SA+5:NEXT

READY.
```

The value $31 loaded into the X register in the BASIC 2.0 version will produce a

?DATA ERROR IN line number zzz

if the target line number does not exist. The error message can be changed by changing the X register value. (see below)

Similarly, in VIC BASIC, the value $18 produces a ?FILE DATA ERROR IN line number zzz

One of the main uses of DATA statements is in filling an array so quick access to data is possible. With a routine such as this, the array, in many cases, is not needed. The number which would normally be used as a subscript to the array is simply used to RESTORE a given line of DATA. The necessary information can then be READ in and used, no waste of time AND a saving of bytes!

Maybe you can think of other uses for a routine such as this. You may wish to add other features or change error messages. Owners of the VIC may wish to use the actual RESTORE command by changing the RAM vectors. EXPERIMENT !

ERROR MESSAGES

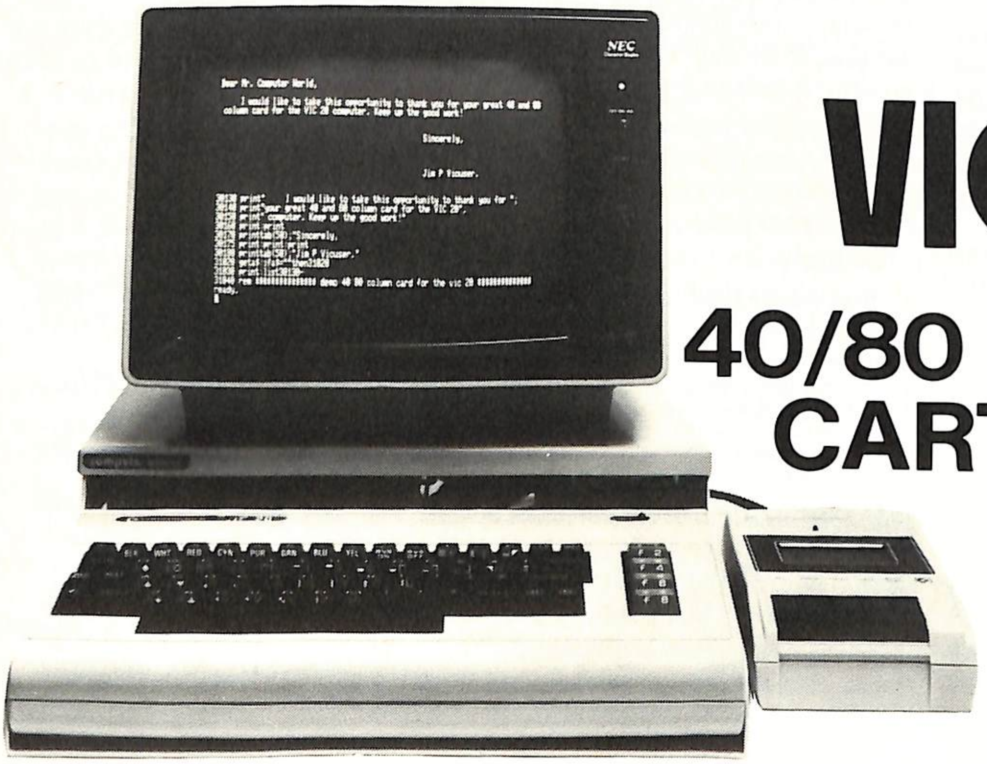
Load the X register with the following values to change the error message.

	BASIC 2.0	VIC BASIC V2
UNDEF'D STATEMENT ERROR	$5A	411
FILE DATA ERROR	$BF	$18
OUT OF DATA ERROR	$2A	$0D
DATA ERROR	$31	—

Due to the different methods of printing error messages on the VIC and the PET, 'DATA ERROR' cannot be obtained on the VIC.

Footnote:
If the target line of the routine is not a DATA line, the next highest DATA line will be RESTOREd. If there is no next DATA line, then an 'OUT OF DATA ERROR' will occur.

This fact may make it undesirable to use 'OUT OF DATA' as an error message if no line exists, as the actual cause of the error may be ambiguous.



VIC=20

40/80 COLUMN CARTRIDGE

- ★ 40/80 Column Display (not colour)
- ★ Vic and CBM Graphics
- ★ Load all PET/CBM Programmes
- ★ All Cursor control Upper and Lower Case
- ★ Switchable from basic without losing programme
- ★ No alterations to Vic required
- ★ No external power supply required
- ★ Works together with 32k Basic Vic computer

CW ELECTRONICS

416 LOGAN RD. (Pacific Hwy.) STONES CORNER,
BRISBANE. TEL: (07) 397 0808, 397 0888. P.O. Box 274
SUNNYBANK QLD. 4109 TELEX AA40811

THE VIC CENTRE

VIC INTERFACE AND PROGRAMMING CARTRIDGES

INTERFACING

The VIC-1011 RS232C ADAPTER is available in two types;

- VIC-1011A TERMINAL TYPE
- VIC-1011B CURRENT LOOP TYPE.

The VIC-1011 RS232C ADAPTER has a standard connector so that it can be used for data communications by being connected directly with an acoustic coupler or a modem.

The VIC RS232 interface is in conformity to the EIA STANDARD (Aug. 1979).

The cartridge is inserted into the user port of the VIC-20 personal computer.

The VIC-1011 RS232C adapter has the following two operation levels:-

1) BASIC level,
2) Bit/Byte handler of machine language level.

BASIC Level

When using the VIC-1011 RS232C adapter at the BASIC level, the standard I/O commands of CBM BASIC (OPEN, CLOSE, CMD, INPUT#, GET#, PRINT#) and the special variable ST are used.

Also, it can be linked with machine language programs for more sophistocated communications and the further expansion of its operating range.

Bit/Byte Handler

RS232C bit/byte level handler operates under the control of VIA (Versatile Interface Adapter), MPS6522 device timers and interrupts. It executes the RS232C protocol while the execution of the BASIC and machine language programs are in progress.

A detailed booklet is supplied with the cartridge to provide information on setting up the cartridge and how to program the VIC-20 to use the RS232C interface.

PROGRAMMERS AID

This cartridge helps new and experienced programmers write, edit, cruch and debug BASIC programs.

This cartridge adds more than 20 special editing commands to the VIC instruction vocabulary and gives an easy method of redefining the VIC's programmable function keys.

If you are an experienced programmer, you already know how helpful it it to automatically number and renumber program lines, delete and insert segments, view program lines as they run, jump to commands or character strings and use the VIC special function keys to increase your programming power.

If you are beginning programming, the Programmers Aid cartridge will make it easier to write programs, understand how they work, and find and correct errors.

SUPER EXPANDER

This cartridge adds an entirely new set of colour, graphics, plotting, music and function key commands to the VIC vocabulary and adds 3K extra memory to load and run larger programs.

Graphic shapes can be plotted, colours 'painted' on the screen, music written and much much more.

The commands and functions work just like BASIC commands. In addition, most of the new commands are pre-assigned to the special function keys.

A valuable cartridge for those who want to develop all the special features of the VIC-20 computer.

MACHINE LANGUAGE MONITOR

When the VIC-20 is turned on, the computer is in 'READY' mode and will accept instructions to run a BASIC program.

BASIC text is converted by the interpreter into machine code instructions which the microprocessor can respond to. If programs are written directly in machine code, faster running speeds will be obtained.

Detailed instructions on machine language programming can be found in the VIC-20 Programmers Reference Guide.

The VIC MON cartridge also allows the user to construct 6502 machine code programs and includes a one line assembler/disassembler.

MEMORY EXPANSION

The VIC-20's user memory can be simple expanded by plugging in one of the memory cartridges.

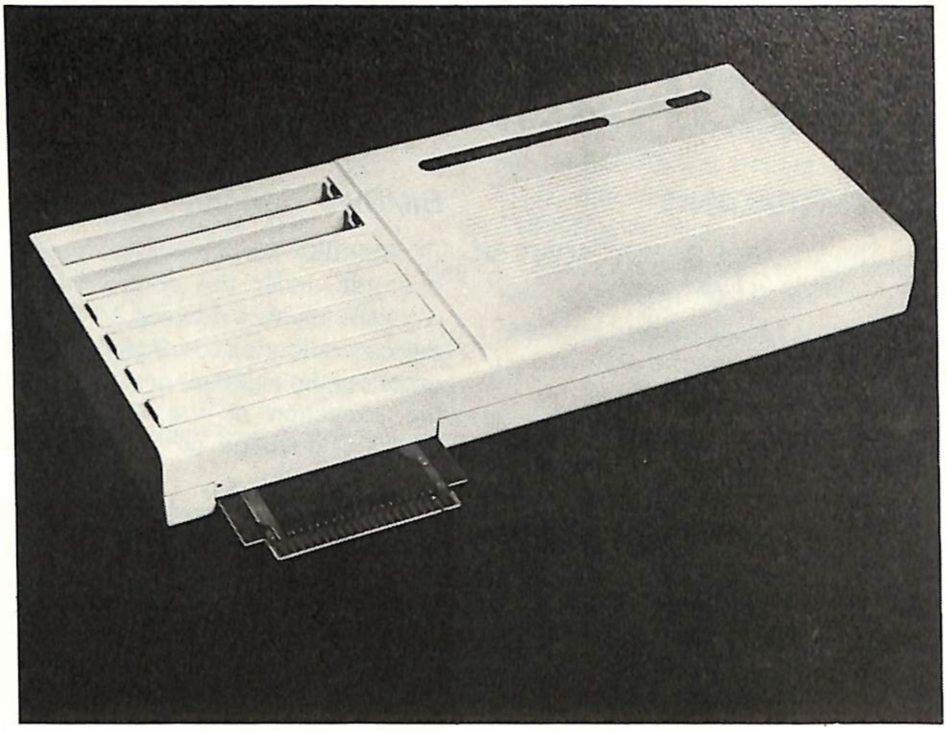
The cartridges are available in 3K, 8K and 16K segments and can be configured to work together when using the VIC 1010 expansion unit.

VIC 1010 MEMORY EXPANSION MODULE

This unit plugs into the memory expansion bus at the rear of the VIC-20 computer.

Six slots are provided to take up to six cartridges such as memory expansion, Programmers Aid, various games etc.

Power is supplied to the VIC 1010 by a mains cable and no other accessories are required to connect to the VIC-20 computer.



SPECIFICATIONS

Input Voltage	**220-240 VAC 50 Hz**	**Dimensions**	**Depth: 235mm (Body 187mm, Edge 48mm)**
Power	**5.05 VDC, 7A**		**Width: 397mm**
Temperature	**0-40 degrees C**		**Height: 53mm**
Humidity	**40-85%**	**Weight**	**1.66Kg**

Letters to the Editor continued from Page 3

I would strongly suggest you contact the Dealer from whom you obtained the package to organise a training session and the benefits will certainly allow you to master wordprocessing.

Mailing lists are described in the training manual under the heading of 'document assembly' and this is detailed in the index.

To explain the paragraph you quoted, the text associated controls are commands that tell the text how to format, such as tab stops, line feeds, indents, underlines etc..

Instead of cluttering the text, they are attached to the appropriate character and the character highlighted in the text by reversing the character (ie a black character on green background). To see what the controls are, then use the 'CONTROL C' function for the line you wish to view, which will show all control characters on lines 3 and 4 on the screen, in the same character position where it is used.

In the example you quoted, the t symbol is a tab stop, indicating a tab point at position 23.

Dear Editor,

How can I obtain VIC schematics and a VIC memory map?

Regards,
Ron Miller.

The memory maps have been published in the last two issues of the Commodore Magazine, however both the schematics, memory maps and other technical information is available in the VIC 20 Programmers Reference Guide which is available from your Commodore Dealer.

The VIC Revealed also has the memory maps and details of the Kernal routines and this book is also available from your Commodore Dealer.

VIC 20 CARTRIDGE DEVELOPMENT

Peter Higginbottom
Commodore Canada

Power-up Activity.

When the VIC 20 is powered on, it looks for a sequence of bytes in a place in memory, which will tell the computer whether or not there is a cartridge plugged in. If this pattern of bytes is not present, the VIC goes about it's usual chores, and comes up with the familiar message:

```
**** CBM BASIC V2 ***
3583 BYTES FREE
            (may be a different amount)
READY.
```

The Sequence of Bytes.

The sequence of bytes the VIC 20 is looking for are located from $A004 to $A008. They should contain:

HEX Address	HEX Contents	CBM ASCII Equivalent
$A004	$41	"a"
$A005	$30	"O"
$A006	$C3	"C"
$A007$C2	·"B"	
$A008	$CD	"M"

To recap then, "a0CBM" is the sequence. If the VIC does see this sequence, what does it do? Well, that's what the first four bytes from $A000 to $A003 are for. If the VIC sees the sequence, it will hump indirectly through $A000, ie. using the vector at $A000 as a power up address. In laymans terms, this means wherever your code is that you want to execute at power-up, the address of it should be stored in "low, high" format at $A000 and $A001. This will be known as the "cartridge cold start vector".

That still leaves $A002 and $A003. They form another vector called the "cartridge warm start vector". This will be vectored through whenever an "NMI" interrupt occurs. For those who are not aware of this, the RESTORE key actually generates an NMI interrupt, and so it may be used to re-start a game, or better still if pressed while holding down another key similtaneously (usually RUN/STOP) to avoid accidental re-starts.

An example in assembly coding might be:

```
 *=$A000
;
.WORD COLD ;LOW, HIGH VECTOR TO
                  FIRST INSTRUCTION
.WORD WARM ;LOW, HIGH VECTOR ON
                  "WARM START"
;
.BYT $41, $30, $C2, $C3, $CD
;
;IGNORE NMI INTERRUPTS
;
WARM PLA
     TAX
     PLA
     TAY
     PLA
     RTI
;
COLD - CARTRIDGE CODE STARTS HERE
;
```

BASIC Programs in a Cartridge.

It is inadvisable to try to "RUN" BASIC programs out of a cartridge. A compromise would be to do the usual power-up activities (ie. test RAM, etc.) and then "move" the BASIC program from the cartridge to RAM, and execute (probably easiest to do by putting a RUN/tRETURN> in the keyboard buffer). It is also inadvisable to put BASIC in a cartridge anyway because of the hazard of someone putting the code onto a more "transportable" medium such as tape or disk. This can be partially avoided by disabling the RUN/STOP key, and the restore key by using the WARM start vector, and changing the IRQ vector into the cartridge.

Cartridge Standards.

Here are some standards that have been laid out for operation of cartridge software:

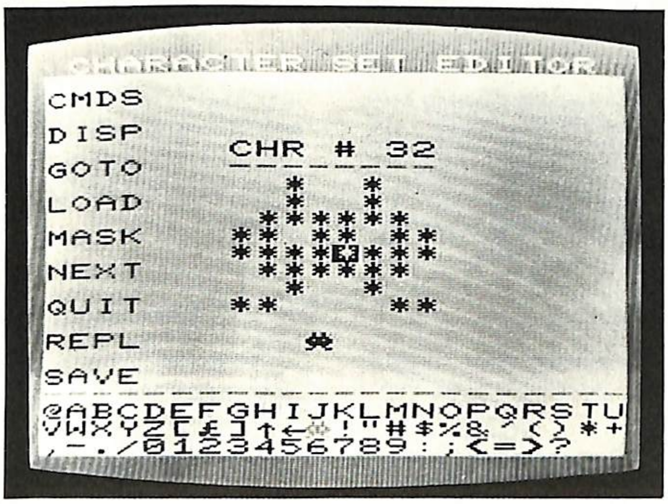
1. Users should have the opportunity to adjust the screen before commencing; horizontal adjust ($9000) using the cursor left/right key and vertical adjust ($9001) using cursor up/down.
2. Where applicable, games should have both keyboard and joystick/paddle control. The keys to use are:

```
                  UP  P
A–'FIRE'     LEFT–L       ; –RIGHT
                 DOWN
```

3. Also where applicable, F1 and F3 are used to choose 1 or 2 players. Afterwards, F1 is used to start play action. Provide about a 5 second delay between "Play Player 1" and "Play Player 2".
4. Function key 7 should flip interlace mode. Some TV's scan every raster "one after another". Other TV's scan every second raster, then go back and do the "in between" rasters. The VIC chip can provide for this by simply setting or unsetting bit 7 of $9000. The user will decide which looks best.
6. Finally, the combination of the RUN/STOP and RESTORE keys should re-boot the game, as if power were interrupted.

USING THE PROGRAMMABLE CHARACTER SET EDITOR

We wanted to tell you about a fascinating new product for the VIC 20 and the best way to do that is to give you an example of what you can do with the product. THE TUMBLER is a screen animation program which you can type directly into your VIC 20 and RUN.



You can use the editor to write the SAME type of program yourself, LIST, SAVE and RUN it in BASIC. Try typing TUMBLER exactly as shown below, and type RUN. You can also SAVE this on tape for future use by using your Commodore DATASSETTE data recorder, or on disk using the VIC 1540 DISK DRIVE.

```
10    REM:TUMBLER BY M. TOMCZYK
11  K=7168:FORJ=32*8TO32*8+7:
                   POKE7168+J,0:NEXTJ
12  READ CH:IFCH=-1THENGOTO20
14  POKEK,CH:K=K+1:GOTO12
20  PRINTCHR$(147):X=64:POKE36869,255
25  PRINT:PRINTSPC(6)CHR$(18)CHR$(30)
         "TUMBLER!"CHR$(146)CHR$(31)
30  POKE36878,15:S=36875:B=128:
                 E=138:PRINTCHR$(31)
40  FORM=BTOESTEP5:NEXTM:POKES,
                         M:POKES,0
50  PRINTTAB(80-X)CHR$(X)CHR$(32)
                          CHR$(145)
60  FORM=BTOESTEP5:NEXTM:POKES,
                         M:POKES,0
70  IFX=79THENEND
80  FORT=1TO150:NEXTT
90  X=X+1:B=B+5:E=E+5:GOTO40
100 FORP=1TO100:NEXTP:GOTO20
500 REM:TUMBLER CHARACTERS
505 DATA24,24,127,24,56,40,40,40
510 DATA96,96,60,50,240,72,36,18
515 DATA96,98,60,16,114,78,66,192
520 DATA50,52,152,113,50,44,32,24
525 DATA0,4,104,113,62,48,73,134
530 DATA65,34,20,217,242,60,192,0
535 DATA65,34,148,89,50,252,192,0
540 DATA48,8,228,28,12,63,48,0
545 DATA64,32,16,240,16,56,84,68
550 DATA0,64,32,24,248,28,42,34
555 DATA1,2,4,254,30,40,72,132
560 DATA38,22,8,24,52,82,144,144
565 DATA12,76,56,14,8,24,36,34
570 DATA48,178,84,56,16,16,40,68
575 DATA177,178,84,56,16,16,40,68
580 DATA48,48,16,56,84,16,16,48,-1
```

The key is the DATA starting on lines 505. What the Character Editor lets us do is this:

1. First we created a series of special characters using the Character Set Editor.
2. We then typed "SAVE" from the editor's command menu and typed "LIST". The list command gave us a list of 8-number DATA statements for each of the special characters we created. Each special character is represented by ONE LINE of DATA statements.
3. The BASIC program we then wrote includes a section which lets us identify our programmable characters as DATA statements, tells the VIC to READ that DATA and create characters in memory and on the screen (POKE36869,255) PRINT those special characters in line 50 (the X is the special character).

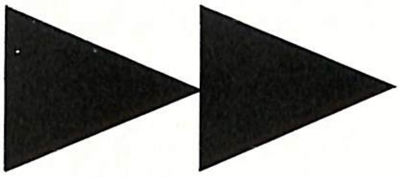
   Immediately after we PRINT the character (CHR$(X)) we PRINT a space to erase that character (CHR$(32)) and tell the VIC to move one space to the left and PRINT the next character one space over. In this way we create the illusion of animation on the screen.
4. When you RUN the program, the programmable characters appear and move. If you want to experiment, you can change some of the numbers in the DATA statements and watch how the characters change.

The VIC Programmable Character Set & Gamegraphics Editor is a useful aid for both BASIC and Machine Language programmers ...and is friendly enough for first-time computerists to use with little or no experience. The Character Editor retails for only $25.00 (Code 100164) and is available from most VIC dealers. It comes with a 16 page instruction manual which talks you through all the features, step-by-step.

The Character Editor lets you perform all sorts of fascinating tricks with your VIC 20 ...but most importantly it lets you create your own characters! This means you can create foreign language character sets, game graphics, special notation symbols ...anything your imagination conjurs up.

FAST FORWARD ON VIC CASSETTE

R.G. Friend,
109 Campbellfield Avenue,
Campbelltown.
N.S.W. 2560.



I am submitting a version of the Fast Foward LOAD/SAVE program which was in your magazine some time ago.

This version is for the VIC 20 and has several changes for the new format screen.

(1) Lines 20 to 110 are allocated for your titles 15 letters max.

(2) The spaces left are to allocate the correct amount of memory.

(3) You can add to and subtract from this programme without changing all the other programmes on the tape.

(4) Space has been left on each of the 10 segments for 8K of memory which will cover at least the 3K expansion.

You may find this of interest to your readers, there were too many changes just to list them so I have submitted a tape which was written on a standard VIC 20.

```
20 FL$(1)="                    |"
30 FL$(2)="                    |"
40 FL$(3)="                    |"
50 FL$(4)="                    |"
60 FL$(5)="                    |"
70 FL$(6)="                    |"
80 FL$(7)="                    |"
90 FL$(8)="                    |"
100 FL$(9)="                    |"
110 FL$(10)="                    |"
140 PRINT"▢▣▣▣PRESS STOP ON TAPE#1"
150 IFPEEK(192)<>0GOTO150
160 PRINT"▢"
170 PRINT"    **  DIRECTORY  ** "
180 PRINT"        ___________  "
190 FORQ=1TO22:PRINT"_";:NEXT:PRINT" "
210 PRINT"| FILE |  DESCRIPTION  |
230 FORQ=1TO22:PRINT"¯";:NEXT:PRINT" "
240 FOR Q=1 TO10:PRINT" | " CHR$(Q+64);" |.";FL$(Q):NEXT
260 FORQ=1TO22:PRINT"¯";:NEXT:
270 PRINT"▣WHICH FILE DO YOU WANT?"
280 GET C$:IF C$="" GOTO 280
290 IF C$<"A" OR C$>"L" GOTO280
300 BS=ASC(C$)-64:F1=BS
310 FT=BS*8.3-8
320 PRINT"▢▣▣▣▣SEARCHING FOR FILE  ";C$:PRINT"▣▣"
330 PRINT"NAMED ";FL$(F1)
340 PRINT"▣▣▣▣▣PRESS FAST FORWARD ON    ▣      TAPE"
350 IF PEEK(37137)<>62GOTO350
360 FT=TI+FT*60
370 IF TI<FT GOTO 370
380 POKE192,52:POKE37148,241
390 PRINT"▢▣▣▣▣▣▣PRESS STOP ON TAPE #1
400 IF PEEK(192)<>0 GOTO 400
405 POKE37148,14
410 PRINT"▢▣▣▣TAPE IS NOW IN CORRECT▣     POSITION FOR"
420 PRINT"▣FILE ";C$;", "FL$(F1)
440 PRINT"▣▣▣   YOU CAN NOW LOAD  ▣       OR SAVE "
450 PRINT"▣       THIS  PROGRAM"
455 END
READY.
```

COMMODORE RANGE OF DISK PERIPHERALS

VIC 1540

Total Capacity 174K.
Low Cost Single disk drive for VIC Computer.
5 1/4" single sided, single density floppy diskettes.

Total Capacity	– 174,848 bytes per diskette
Sequential	– 168,656 bytes per diskette
Relative	– 167,132 bytes per diskette
Records/file	– 65,535
Directory Entries	– 144 per diskette
Bytes per sector	– 256
Sectors	– 17-21
Tracks	– 35
Blocks	– 683 per diskette (644 free)

CBM 2031

Total Capacity 174K.
Low Cost Single disk drive.
5 1/4" single sided, single density floppy diskettes.

Total Capacity	– 174,848 bytes per diskette
Sequential	– 168,656 bytes per diskette
Relative	– 167,132 bytes per diskette
Records/file	– 65,535
Directory Entries	– 144 per diskette
Bytes per sector	– 256
Sectors	– 17-21
Tracks	– 35
Blocks	– 664 per diskette

CBM 4040

Total Capacity 340K.
Dual Disk Drive.
5 1/4" single sided, single density floppy diskettes.

Total Capacity	– 174,848 bytes per diskette
Sequential	– 168,656 bytes per diskette
Relative	– 167,132 bytes per diskette
Records/file	– 65,535
Directory Entries	– 144 per diskette
Bytes per sector	– 256
Sectors	– 17-21
Tracks	– 35
Blocks	– 664 per diskette

CBM 8050

Total Capacity 1,060K.
Dual Disk Drive.
5 1/4" single sided, single density floppy diskettes.

Total Capacity	– 533,248 bytes per diskette
Sequential	– 521,208 bytes per diskette
Relative	– 464,312 – 517,398 bytes per diskette
Records/file	– 65,535
Directory Entries	– 224 per diskette
Bytes per sector	– 256
Sectors	– 23-29
Tracks	– 77
Blocks	– 2083 per diskette

2CBM 8250

Total Capacity 2,100K.
Dual Disk Drive.
5 1/4" double sided, single density floppy diskettes.

Total Capacity	– 533,248 bytes per diskette
Sequential	– 521,208 bytes per diskette
Relative	– 464,312 – 517,398 bytes per diskette
Directory Entries	– 224 per diskette
Bytes per sector	– 256
Sectors	– 23-29
Tracks	– 77
Blocks	– 2083 per diskette

CBM 9060

Total Capacity 5,000K.
Hard Disk Drive.

Total Capacity	– 6,380,000 unformatted
	– 4,970,000 formatted
Sequential	– full disk capacity
Relative	– full disk capacity
Bytes per sector	– 256
Sectors	– 32
Recording Surfaces	– 4
Tracks	– 153 per recording surface
Blocks	– 19,442

CBM 9090

Total Capacity 7,500K.
Hard Disk Drive.

Total Capacity	– 9,570,000 unformatted
	– 7,460,000 formatted
Sequential	– full disk capacity
Relative	– full disk capacity
Bytes per sector	– 256
Sectors/Track	– 32
Recording Surfaces	– 6
Cylinders(Tracks)	– 153 per recording surface
Blocks	– 29,162

Continued on Page 32

upd MicroPro Design Pty. Ltd.
Specialising in the sales & support of the Commodore PET/CBM Microcomputers, peripherals and interfaces, including:

- IEEE488-RS232 COMMUNICATIONS
- RS232/CENTRONICS PRINTER INTERFACE
- EPROM PROGRAMMER
- WORD PROCESSOR PRINTER INTERFACE

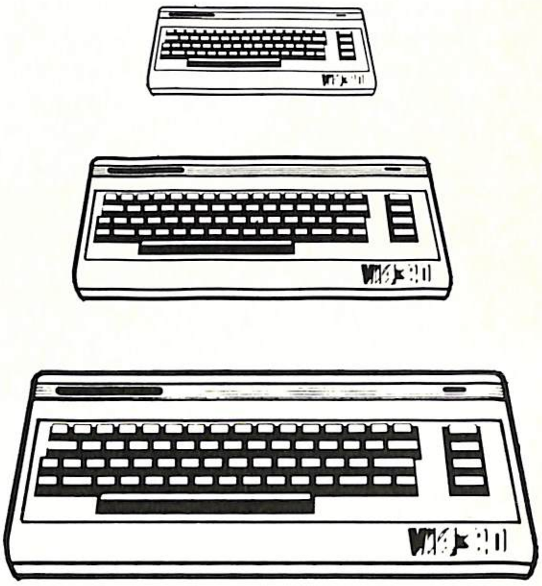
For full details and prices call or write:
205/6 Clarke St
CROWS NEST
Ph:(02) 438 1220
Postal: P.O. Box 153
NORTH SYDNEY
NSW 2060

M B G
PHOTOTYPESETTING
★ Phototypesetting from text disks and/or cassettes generated from Commodore microcomputers.
★ Complete art studio facilities
★ Reports, software manuals, advertising, etc. etc.
(02) 439 1827
MERVYN BEAMISH GRAPHICS
82 Alexander St. CROWS NEST, NSW 2065

USING THE VIC EXPANSION MODULE

by Garry Mason

Many people have purchased the cassette games such as Blue-Meanies, Car Chase, etc have had difficulty in running 3K games on VIC's fitted with 8K/16K RAM packs. This is due to the way the VIC is initialised when the power is turned on. This determines the way in which memory space is allocated to the screen and colour memory, as well as the initialisation of all the appropriate pointers such as Start of BASIC text.



When the power is first turned on, the reset line on the 6502 chip maintains the VIC in a reset state for a few seconds. This then forces the start of a routine at location $FFFC and $FFFD in memory. This then instructs the VIC to start a 'cold start' program located at $FD22. The operating system is set up so that the BASIC interpreter can then take over. The first thing this routine does is to check to see if there is a games cartridge installed by looking at 5 locations from location $A004 for a sequence of characters, "aoCBM".

If this sequence is present, then the VIC will jump to a program that is pointed to by the contents of locations $A002 and $A003. Otherwise the VIC will proceed to set up the VIC chip, the operating system variables and intialise it. From here on in the system has to know what combination of RAM is being used and once this has been done the BASIC Interpreter can be set up.

If there is any RAM found in locations $0400 to $1000 and none found at $2000 then –

Set Start of BASIC to $0400 or 1024, with the screen RAM to start at $1E00 or 7680 and the colour RAM to begin at location $9600 or 38400. This is the case with either the 3K RAM or the Super Expander.

If there is memory from $2000 with an 8K or 16K cartridge inserted then –

Set Start of BASIC at $1200 or 4608, with the screen RAM to start at $1000 or 4096 and the colour RAM will be set up to $9400 or 37888. With this system any memory not in a continuious flow from cartridge to cartridge will not be mapped in the system. This will include the 3K RAM cartridge and the super expander.

In all other cases (no RAM at $4001 or $2001) the system is set up as follows –

Set Start of BASIC to $1000 or 4096, with the screen RAM at $1E00 or 7680 and the colour RAM in this case set up to start at $9600 or 38400.

These movements of pointers is the reason why the VIC cassette software will produce very different results than desired with expansion over $2000.

The Start of BASIC text pointer has also moved, but KERNAL corrects for this automatically when LOADing or SAVEing programs, so that BASIC programs will always LOAD into the correct place depending on what configuration the system is in.

As you may know hi-resolution graphics can be accomplished on the VIC by moving the character generator into RAM as mentioned in the previous issues of the Commodore Magazine. This is restricted to the VIC's internal RAM 4096 – 8192. The screen must reside here also. The reason for this is the restrictions in the addressing capabilities of the VIC chip. If we refer to the memory map for expansion over $2000, we can see that there is no room for any high resolution graphics areas.

In order to use a program that is written for a standard VIC that includes either High Resolution graphics or a redefinded character set, the following programs can be run. These programs should be run before any other program. Basically what the routine does is to reset the machine so that the screen and other pieces of information are put back into the position that would be used by a VIC with only a 3K RAM Pack in place. So now the screen starts at the normal location starting at $1200 and BASIC starts at $2000. If the correction program is run it will free 3.5K of RAM – enough for a 20 x 22 character size screen, giving 160 x 176 pixels of resolution 28160 points. This is the same as that used by the super expander.

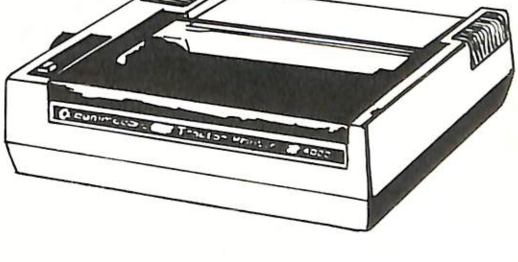
Lines 50 and 55 of the first program will automatically load a file called "prg" from disk. The command string is contained in a$ on line 50 in program 2) and can easily be changed to any ten character command. The only modification needed to the original program is to check for POKE 56 and 52 and remove them. This is because the values that would be put in these locations would not apply to the new set up. They are also no longer required as there is now no need to protect the graphics area, as it is no longer contained in part of the BASIC working space. Do not try incorporating this program into the original program as this will not work, as a 'warm start' is done to set all the parameters.

```
PROGRAM NUMBER 1).


 10 POKE 642,32 :REM CHANGE START OF MEMORY
 20 POKE 648,30 :REM SET SCREEN MEMORY PAGE TO $1E00
 30 POKE36869,240:REM RESET SCREEN & CHR. MAP POINTERS
 40 POKE 36866,PEEK(36866)OR 128 : REM FIX OTHER BIT TO SCREEN POINTER
 50 A$="L┌"+CHR$(34)+"PRG"+CHR$(34)+",8":A=LEN(A$)
 55 POKE198,0:FORC=1TOA :POKE 630+C,ASC(MID$(A$,C,1)):NEXTC:
POKE630+1+C,13:POKE 198,A+1
 60 POKE0,108 :POKE1,0 :POKE2,192: SYS 0 :REM DO BASIC COLD START
READY.
```

```
PROGRAM NUMBER 2).


 10 POKE 642,32 :REM CHANGE START OF MEMORY
 20 POKE 648,30 :REM SET SCREEN MEMORY PAGE TO $1E00
 30 POKE36869,240:REM RESET SCREEN & CHR. MAP POINTERS
 40 POKE 36866,PEEK(36866)OR 128 : REM FIX OTHER BIT TO SCREEN POINTER
 50 POKE0,108 :POKE1,0 :POKE2,192: SYS 0 :REM DO BASIC COLD START
READY.
```

4022 PRINTER NOTES

First I'd like to mention that a new ROM is available for the 3022 printer. The 901472-07 (Commodore part number 931172), replaces the original -03 ROM and the subsequent -04. Recall, the -03 forced a carriage travel for each line feed and the -04 would occasionally lock into lower case. The -07 fixes all previous bugs, but due to 3022 mechanics it can't give bi-directional print. For details on the -07, contact your nearest Commodore Dealer.



Now on to the 4022. Faster, sleeker, quieter, a nicer character set, plus all the features of the 3022 make the 4022 a better printer. The 4022P has since replaced the 4022. It has all the features of it's predecessor with bi-directional print capability added. This new ROM for 4022 printers to allow bi-directional printing, is available for retrofitting. However, like the 3022 printer, Commodore had two goes at getting it right. The first attempt was 325301-01, but had the problems of paging and print formatting and was successfully replaced with a 901631-02 (Commodore part number 931173).

The manual has a few minor oversights which we'll clear up now:

1) Page 31 states, "... 144 steps per inch, so a declared value of 18 produces 8 lines per inch." It should be read, "... 195 steps per inch, ...28 produces 8 lines per inch."

2) Page 31 also says, "PRINT#6,CHR$(144) produces lines spaced one inch apart." Secondary address 65 does not accept values over 127. Therefore CHR$(127) will result in maximum line spacing (approx. 1/2 inch).

3) Next it says, "default value is 24 for the standard 6 lines per inch." Change this to 36.

Here are some line spacing values for secondary address 6. (lpi = lines per inch):

CHR$ Value	Result		Comment
1	195	1pi	;characters will overlap
7	25	1pi	; characters still overlap
12	16.6	1pi	
14	14.2	1pi	
21	10	1pi	;characters stop overlapping
28	8	1pi	
36	6	1pi	
64	5	1pi	
66	4	1pi	
97	3	1pi	

The only other difference we've found occurs when "skipping" from one formatted field to the next. Like the 3022, the 4022(P) supports 'printing data according to a previously defined format'. In order to "skip" from one field to the beginning of the next, it was necessary to send a CHR$(29), or 'Cursor-Right'. This still applies to alpha fields, but when sending numerics to secondary address 1 on the 4022, the skip character is no longer needed. It seems that the 4022 recognises the cursor-right that the PET/CBM tags to the end of numerics and numeric variables when output. For example:

```
3022: PRINT#1,A:CHR$(29);B;CHR$(29);C
4022: PRINT#1,A;B;C
```

If the cursor-rights are sent to the 4022, the printer will skip two fields instead of just one.

IDENTIFYING THE DISK DRIVE

Frequently it is necessary and/or desirable for a program to determine the type of disk system that is operating, without intervention by the operator.

The following ideas can be incorporated into your program to tell which disk drive is being used.

To read a byte from the disk memory, use the following code. Further information on the 'MEMORY-READ' instruction is available in the DOS manual page 52.

```
10 OPEN 15,8,15                  ;Open secondary address 15
20 PRINT#15,"M-R"CHR$(254)CHR$(255)
                                 ;Read value from $FFFE
30 GET#15,A$                     ;Get value as A$
40 A=ASC(A$)                     ;A equals ASCII value
```

The following chart shows the ASCII equivalent of the byte that is located at address $FFFE in the indicated disk system. This may be used to differentiate the disk drive being used.

DISK SYSTEM	BYTE AT $FFFE	
	HEX	ASCII
2040	D6	214
4040<1 JAN 82	OB	11
4040>1 JAN 82	OA	10
8050 REGULAR	E9	233
8050 W/SUPER DOS	11	17 NOTE 1
8250	11	17 NOTE 1
9060/9090	36	54 NOTE 2
2031	B6	182

NOTE 1:
In 8050/8250 systems that are equipped with the "SuperDOS", the system type may be determined by reading address $10AC. An 8050 will return a '1', and 8250 will return a '2'.

NOTE 2:
9060/9090 systems may be differentiated by reading address $109D. A 9060 will return a '4', a 9090 will return a '6'.

SOFTWARE CONTROLLED IMMEDIATE MODE.

By Dirk Williams.

Most people who are familiar with Commodore computers, will realise that there are two basic modes of computer operation:-

1) Immediate mode – from which programs are entered, edited, saved or loaded, etc.
2) Program mode – where the computer is under the control of the currently residing program.

Think how nice it would be if one could combine the two modes, thus allowing programs to write themselves, edit themselves, etc during execution. Thanks to the keyboard input buffer contained in the BASIC workspace in all machines, this is entirely possible. The keyboard input buffer is used to store the values of any keypresses, before they are serviced. If you use the keyboard from immediate mode their corresponding service functions are executed almost immediately. Service functions may vary from printing a character at the current cursor position to clear screen, etc. You may have that if you use the keyboard during program execution (ie. – press CLR/HOME) – the corresponding function will be serviced when the program finishes and returns you to immediate mode (ie. - the screen will clear). Note that GET and INPUT statements use the buffer, so any keypresses stored at the time of their execution will be removed. You may also have noted that a program line may be entered, or the program may be RUN simply by pressing return (executing a CHR$(13)) when the cursor is on the same line (screen) as the abovementioned.

To achieve our aim we must firstly print onto the screen any lines or commands we wish to be entered, or executed. Next we must make the computer think that we have pressed return by inserting the keyvalue into the input buffer. When the computer moves back into immediate mode it will service the keyboard buffer, as though you had pressed return – and will enter any lines or commands, onto the screen line, where the cursor is at the time. The buffer is 10 characters long, but this can be altered on the CBM 8032 and the VIC 20. There is also a location in zero page that contains the number of keystrokes stored in the buffer. The locations are as follows:-

POINTER TO NUMBER OF KEYSTROKES IN BUFFER:

BASIC 4 –	$009E	158(decimal)
BASIC 2 –	$009E	158(decimal)
VIC 20 –	$00C6	198(decimal)

POINTER TO START OF KEYBOARD BUFFER

BASIC 4 –	$026F	623(decimal)
BASIC 2 –	$026F	623(decimal)
VIC 20 –	$0277	631(decimal)

POINTER TO LENGTH OF KEYBOARD BUFFER

CBM 8032 –	$00E3	227(decimal)
VIC 20 –	$0289	649(decimal)

The following program demonstrates the various methods involved in the technique. The program is a cassette based address book, which writes data statements into which the addresses are inserted.

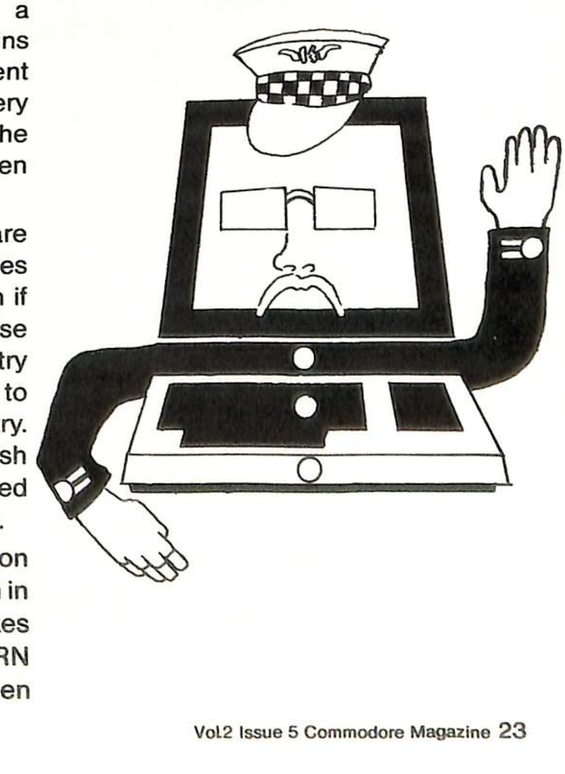
Lines 1000 onward contain the data statements. Line 1000 is a pointer DATA statement that contains the current highest DATA statement line number – it is updated after every entry. Note that you must re-save the program if any entries have been made (main menu option 3).

Lines 500 – 570 and 900 – 920 are used as an input routine that uses GET statements and will not crash if you inadvertantly press return. Use cursor-up to move to a previous entry field when entering – DEL key to delete – CLR/HOME to abort entry. POKE 167,0 makes the cursor flash during entries – this must be restored to 1 (POKE 167,1) to stop flashing.

Lines 800 to 840 prints the lines on the screen as if you had typed them in – followed by a RUN. Line 835 makes the computer think that 6 RETURN keypresses have taken place. When the program is stopped in line 840, the computer will input the lines as though you had pressed return when the cursor is on each of the lines. When it comes to the RUN which has been printed on the screen – the computer will run the program again. If in your programs you wish the program to continue running from a specific line number – then print RUNxxx (where xxx is the line number) instead of just RUN. Note that by pokeing 158,7 in line 835 we tell the computer that there are 7 keypresses to service – it is merely enough to fill the buffer with carriage returns.

Line 190 demonstrates a different approach in that the letters R, V and N followed by a carriage return (chr$(13)) are put into the keyboard buffer – so that the whole word RUN will appear typed in – and will be serviced after the program has SAVED itself.

I hope this has made things a little clearer, if not, well – you still have a handy program to use.



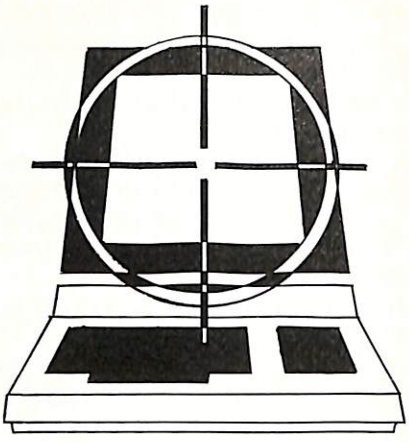
```
5 REM ### BY DIRK WILLIAMS ###
10 CLR:DIMX$(5):PRINT"⊐    CASSETTE ADDRESS FILE"
20 PRINT"◙◙◙◙◙◙             ▧ 1 ■...RECALL":PRINT
30 PRINT"           ▧ 2 ■...ADD":PRINT
40 PRINT"           ▧ 3 ■...RESAVE PROGRAMME":PRINT
50 PRINT"           ▧ 4 ■...ESCAPE":PRINT
60 PRINT"◙◙◙       INPUT OPTION NUMBER: ";
70 POKE167,0
80 GETH$:IFH$<>"1"ANDH$<>"2"ANDH$<>"3"ANDH$<>"4"THEN80
85 POKE167,1
90 PRINTH$:IFH$<>"4"THEN180
100 PRINT"⊐◙◙       ENSURE THAT PROGRAMME HAS BEEN"
110 PRINT"      RECSAVED IF ANY ADDITIONS HAVE"
120 PRINT"      BEEN MADE DURING USE"
130 PRINT"◙◙       PRESS ▧RETURN■ TO RETURN TO MENU"
140 PRINT"      PRESS ANY OTHER KEY TO ESCAPE"
150 GETH$:IFH$=""THEN150
160 IFH$=CHR$(13)THEN10
170 U:SYS64790
180 IFH$<>"3"THEN210
190 POKE158,4:POKE623,82:POKE624,85:POKE625,78:POKE626,13
200 PRINT"⊐":SAVE"ADDRESS FILE":END
210 IFH$<>"1"THEN600
220 PRINT"⊐           RECALL OPTION MENU"
230 PRINT"◙◙◙◙          ▧ 1 ■...NAME":PRINT
240 PRINT"         ▧ 2 ■...ADDRESS (STREET)":PRINT
250 PRINT"         ▧ 3 ■...SUBURB":PRINT
260 PRINT"         ▧ 4 ■...POST CODE-(STATE)":PRINT
270 PRINT"         ▧ 5 ■...TELEPHONE NUMBER":PRINT
280 PRINT"         ▧ 6 ■...RETURN TO MAIN MENU":PRINT
290 PRINT"◙◙◙       INPUT OPTION NUMBER: ";
300 POKE167,0
310 GETH$:IFH$<>"1"ANDH$<>"2"ANDH$<>"3"ANDH$<>"4"ANDH$<>"5"ANDH$<>"6"THEN310
315 POKE167,1:T=VAL(H$)
320 IFH$="6"THEN10
330 RESTORE:READL:IFL=1000THEN10
340 L=(L-1000)/10:C=0
341 IFT=1THENDT$="NAME"
342 IFT=2THENDT$="ADDRESS"
343 IFT=3THENDT$="SUBURB"
344 IFT=4THENDT$="POST CODE"
345 IFT=5THENDT$="TELEPHONE#"
350 PRINT"⊐◙◙◙◙◙           INPUT ";DT$;" OR PART:"
360 IFH$="1"ORH$="2"THENZ=29
365 IFH$="3"THENZ=15
366 IFH$="5"THENZ=15
367 IFH$="4"THENZ=20
370 RT$="":N=0:X=1:Y=7:GOSUB500:IFH$="▩"THEN220
380 PRINT"⊐              SCANNING RECORDS◙◙◙"
385 PRINT"   ┌──────────────────────────────────────┐ "
386 FORG=1TO6
387 PRINT"   | ";TAB(38);"| ":NEXT
388 PRINT"   └──────────────────────────────────────┘ "
389 PRINT"◙◙◙◙◙◙◙◙◙◙◙  ADDRESS FILE - RECORD RECALL OPTION"
390 FORK=1TOL
410 FORQ=1TO5:READX$(Q):NEXT
430 Y$=LEFT$(X$(T),LEN(RT$))
440 IFY$=RT$THEN450
```

```
386 FORG=1TO6
387 PRINT"   | ";TAB(38);"| ":NEXT
388 PRINT"   └──────────────────────────────────────┘ "
389 PRINT"◙◙◙◙◙◙◙◙◙◙◙  ADDRESS FILE - RECORD RECALL OPTION"
390 FORK=1TOL
410 FORQ=1TO5:READX$(Q):NEXT
430 Y$=LEFT$(X$(T),LEN(RT$))
440 IFY$=RT$THEN450
```

```
441 NEXT:IFC=0THENX=10:Y=7:GOSUB900:PRINT"RECORD DOES NOT EXIST";CHR$(7)
```

```
442 PRINT"⌂                FINISHED !!";CHR$(16);CHR$(7)
443 PRINT"⌂↓       PRESS ANY KEY WHEN READY"
444 GETH$:IFH$=""THEN444
445 GOTO220
450 FORA=1TO5:X=5:Y=5+A:GOSUB900:PRINTX$(A);:GOSUB920:NEXT:C=C+1
460 GETC$
470 X=8:Y=17:GOSUB900:PRINT"PRESS ®SPACE■ TO CONTINUE":FORTT=0TO200:NEXT
471 IFC$=" "THENGOSUB900:PRINTCHR$(16):GOTO441
475 X=8:Y=17:GOSUB900:PRINT"PRESS SPACE TO CONTINUE":FORTT=0TO200:NEXT
478 IFC$=" "THENGOSUB900:PRINTCHR$(16):GOTO441
480 GOTO460
500 Y=Y+1:GOSUB900:FORV=1TOZ:PRINT"¯";:NEXT:Y=Y-1:GOSUB900:POKE167,0
505 PRINTRT$;CHR$(16);
510 GETH$:IFH$=""ORH$="↓"ORH$="→"ORH$="↑"ORH$="←"THEN510
515 IFN=1ANDH$="♥"THENPOKE167,1:RETURN
520 IFH$="⌂"THENPOKE167,1:RETURN
540 IFH$=CHR$(20)ANDLEN(RT$)>0THENPRINTH$;:RT$=LEFT$(RT$,LEN(RT$)-1):GOTO510
550 IFH$=CHR$(13)ANDLEN(RT$)>0THENPOKE167,1:RETURN
560 IFLEN(RT$)+1>ZTHENPRINTCHR$(7);:GOTO510
565 IFH$=CHR$(20)ORH$=CHR$(13)ORH$="♥"ORH$="↓"ORH$="→"ORH$="↑"THEN510
570 RT$=RT$+H$:PRINTH$;:GOTO510
600 RESTORE:READL:L1=(L-1000)/10:L=L+10
610 PRINT"♥    ADD RECORD - THERE ARE ";L1;"RECORDS"
620 PRINT"↓↓↓       NAME:↓"
630 PRINT"   ADDRESS:↓"
640 PRINT"    SUBURB:↓"
650 PRINT" POST CODE:↓"
660 PRINT"TELEPHONE#:↓"
670 RT$=X$(1):N=0:X=11:Y=4:Z=29:GOSUB500
675 IFH$="⌂"THEN10
676 X$(1)=RT$
680 RT$=X$(2):N=1:X=11:Y=6:Z=29:GOSUB500
685 IFH$="⌂"THEN10
686 X$(2)=RT$
687 IFH$="♥"THEN670
690 RT$=X$(3):N=1:X=11:Y=8:Z=15:GOSUB500
695 IFH$="⌂"THEN10
696 X$(3)=RT$
697 IFH$="♥"THEN680
700 RT$=X$(4):N=1:X=11:Y=10:Z=20:GOSUB500
705 IFH$="⌂"THEN10
706 X$(4)=RT$
707 IFH$="♥"THEN690
710 RT$=X$(5):N=1:X=11:Y=12:Z=15:GOSUB500
715 IFH$="⌂"THEN10
716 X$(5)=RT$
717 IFH$="♥"THEN700
718 PRINT
720 PRINTX=11:Y=20:GOSUB900:PRINT"ARE YOU SURE ?";:POKE167,0
730 GETH$:IFH$<>"Y"ANDH$<>"N"ANDH$<>"⌂"THEN730
731 POKE167,1:IFH$="⌂"THEN10
732 IFH$="N"THENGOSUB900:PRINTCHR$(16):GOTO670
800 RESTORE:READL:L=L+10:PRINT"♥↓↓↓↓"
810 PRINTL;"DATA ";X$(1)
811 PRINTL+2;"DATA ";X$(2)
813 PRINTL+4;"DATA ";X$(3)
814 PRINTL+6;"DATA ";X$(4)
815 PRINTL+8;"DATA ";X$(5)
816 PRINT"1000DATA";L
817 PRINT"RUN"
830 PRINT"⌂↓↓"
835 POKE158,7:FORT=0TO7:POKE623+T,13:NEXT:END
840 STOP
900 PRINT"⌂";:FORS=1TOX:PRINT"→";:NEXT:FORS=1TOY:PRINT"↓";:NEXT:RETURN
920 FORLF=1TO30-LEN(X$(A)):PRINT" ";:NEXT:PRINT" ":RETURN
1000 DATA 1000
READY.
```

MACHINE LANGUAGE AUTO-LOCATION



When a program like Supermon or Tinymon loads into its computer and RUN is given, it builds a copy of the "real" program in high memory. There's a need to do this: different computers have different memory sizes, and we want to find the top of memory wherever it is. Also, the computer might already have something also near the top of memory (such as a wedge program) and we want the new program to fit neatly below it.

This calls for an auto-location program. The object program must be packed into high memory. This is often more than just moving the program, since some things may need to be changed with the move. If you have a program that uses only branches – no jumps, no in-program subroutines, no tables – you may be able to get away with a simple move operation. But any instruction that uses an in-program absolute address: jumps, subroutine calls, and tables – will need to be adjusted.

We need to build a relocatable program module. Something that says, "This byte is normal so we may just move it; but that pair of bytes is an address and must be recalculated for the new location."

Ground Rules.

We need a scheme which marks addresses so that the proper arithmetic may be performed. There's one requirement as to how you write the program: it may be summarized as "all addresses must be in one piece"...

The rule makes sense: it would be difficult to perform arithmetic on an address whose two bytes were scattered in different parts of the program. For users with assemblers, the rule translates to: never use the < or *gt* functions for high and low byte.

So if we wanted to place the address of TABLE into indirect address INDAD, we would avoid coding: LDA #<TABLE : STA INDAD : LDA #>TABLE : STA INDAD+1. Instead, we'd define the table address in memory with TABLAD.WORD TABLE and perform the above setup with LDA TABLAD : STA INDAD : LDA TABLAD+1 : STA INDAD+1. We've used four more bytes but gained a major benefit: the two bytes representing the address of TABLE are now stored together (at TABLAD) and we can adjust this address easily when we wish to relocate.

The Method.

The way we build a relocatable module is quite easy. Any time we see an address that will need relocation, we place a zero above it. As we repack the program (from the top down) the zero will signal that a relocatable address follows. That's all very well, but what do we do with real zeros? There will be many zeros in the program itself, and we don't want them to trigger a false relocation calculation. In this case, we change the zero to two zeros in the relocatable package. The relocation program will spot this and change it back to a single zero. In order to do arithmetic on the addresses, we need to know where they are pointed in the first place. To relocate from $1000 to $4000, for example, we need to add $3000; but we must know that we are starting from $1000. I use the following convention: adresses are written so that the top of the program plus one is $0000 – that is, the last byte of the relocatable program is $FFFF. The program can't really go there, since that's ROM space, but it makes the arithmetic easy. We can look at an address in the relocation package as a signed number: address $FFC0 can be viewed as "64 bytes from the top of the program." If our real top-of-program turned out to be $8000, which would be correct for a 32K machine, we would translate the sequence 20 C0 FF 00 to 20 C0 7F ...note that the zero disappears; it's the relocation flag. How did we get the new address $7FC0? By adding the relocation address, $FFC0, to the top-of-program, $8000.

Generating the Relocatable Program.

How do we manufacture this package with zeros added and addresses recalculated, ready for relocation? With an assembler it's quite easy.

First, we assemble two versions of the program at two different locations. That's easy enough to do: we just change the *= statement at the start of our source code.

Then we run a simple compare program which compares the two object programs we have assembled, starting from the top. Each matching byte is copied into the relocation area unchanged; if it's a zero, an extra zero is added. If the bytes don't match, we have a relocatable address: in this case, we insert the zero plus the recalculated address into the relocation package. It's an easy job: my "relocate builder" is a BASIC program of about a dozen lines.

Stopping.

As we work down from the top we need to detect when we have reached the end of the program: this is true of both the relocate builder and the relocating program itself. There are many easy ways of doing it. The program can test to see if the last address has been reached. Alternatively, we can put some sort of "flag" into the coding itself to detect the end. In TINYMON, I use a value $BF which is never used in the program, as a simple detection. A more complete method might be to use a zero with a value of 1 stored below it. It's up to you: whatever works is OK.

VIC Note.

In the VIC, we have one more problem to solve. We can find the top of memory (locations $37 and $38) but our program might fall into different memory space, depending what's plugged in. Use pointers to find your own program (try $2D and $2E) and everything should work out nicely.

Summary.

You can pick apart the code of SUPERMON or TINYMON and see how it's done. You can develop your own programs. But if you understand the principles of a relocating program package, you can develop significantly more useful programs which will adapt to a wider variety of machine configurations.

Editors Note.

The machine code disassembly to follow is Jim Butterfield's relocator modified slightly by Dave Hook for use with his VICloader for PET/CBMs. Dave eliminated the JMPs and JSRs in Jim's original utility so that the relocator can be relocated. For VICloader, it starts at $0640, but you can move it anywhere; higher if you want more BASIC underneath it, or lower for larger object programs.

Notice that the relocator starts with the end of the object program since this will be the first byte to be packed into high memory. This is conveniently pointed at by the Start of Variables pointer minus 1, which is set on completion of the LOAD (provided it is saved properly).

```
0400-063F    BASIC portion (title, sys address, etc)

0640 A5 2A     LDA $2A       ;store copy of
0642 85 1F     STA $1F       ;Start of Variables
0644 A5 2B     LDA $2B       ;pointer (last byte of
0646 85 20     STA $20       ;object program + 1).
0648 A5 34     LDA $34       ;store copy of
064A 85 21     STA $21       ;Top of Memory
064C A5 35     LDA #35       ;pointer (MemTop)
064E 85 22     STA $22
0650 A0 00     LDY #$00      ;zeroise Y index
0652 A5 1F     LDA $1F       ;dec pointer to last
0654 D0 02     BNE $0658     ;byte of object prog.
0656 C6 20     DEC $20       ;(1st byte to be
0658 C6 1F     DEC $1F       ;packed)
065A B1 1F     LDA ($1F),Y   ;get obj. prog. byte
065C D0 3C     BNE $069A     ;not 0, goto $069A
065E A5 1F     LDA $1F       ;if 0, dec pointer
0660 D0 02     BNE $0664
```

→

Continued from page 27

```
0662 C6 20      DEC $20
0664 C6 1F      DEC $1F
0666 B1 1F      LDA ($1F),Y     ;and get next byte
0668 F0 21      BEQ $068B       ;0? yes, true zero *
066A 85 23      STA $23         ;no, relocatable addr
066C A5 1F      LDA $1F         ;store high byte in
066E D0 02      BNE $0672       ;$23.  dec pointer
0670 C6 20      DEC $20         ;and
0672 C6 1F      DEC $1F
0674 B1 1F      LDA ($1F),Y     ;get next byte
0676 18         CLC             ;recalculate lo addr
0677 65 21      ADC $21         ;using MemTop lo
0679 AA         TAX             ;result in .X
067A A5 23      LDA $23         ;recalculate hi addr
067C 65 22      ADC $22         ;using MemTop hi
067E 48         PHA             ;result on stack
067F A5 34      LDA $34         ;dec MemTop
0681 D0 02      BNE $0685
0683 C6 35      DEC $35
0685 C6 34      DEC $34
0687 68         PLA             ;retrieve hi addr
0688 91 34      STA ($34),Y     ;pack at ($Memtop) .Y=0
068A 8A         TXA             ;retrieve lo addr
068B 48         PHA             ;* save on stack
068C A5 34      LDA $34         ;dec MemTop
068E D0 02      BNE $0692
0690 C6 35      DEC $35
0692 C6 34      DEC $34
0694 68         PLA             ;retrieve byte
0695 91 34      STA ($34),Y     ;pack at ($MemTop) .Y=0
0697 18         CLC             ;rather than
0698 90 B6      BCC $0650       ;a JMP
069A C9 BF      CMP #$BF        ;last byte?
069C D0 ED      BNE $068B       ;no, goto $068B *
069E A5 34      LDA $34         ;yes, set
06A0 85 30      STA $30         ;Bottom of Strings
06A2 A5 35      LDA $35         ;= MemTop
06A4 85 31      STA $31         ;pointer
06A6 6C 34 00   JMP ($0034)     ;jmp to program
06A9 BF                         ;end detector of obj prog
06AA ...                        ;start of object prog
```

DISK UTILITIES

```
     DISK MEMORY DISPLAY - DOS 2.0

100 PRINT"]▮DISK MEMORY DISPLAY     JIM BUTTERFIELD"
105 L=8:IFPEEK(32848)=4THENL=16
110 DATA77,45,87,0,18,16,162,0,189
120 DATA157,64,06,232,224,16,208,245,108,2,252
130 FORJ=1TO9:READX:C$=C$+CHR$(X):NEXTJ
140 FORJ=1TO11:READX:D$=D$+CHR$(X):NEXTJ
150 PRINT"▮THERE ARE TWO PROCESSORS:"
160 PRINT"  1) THE IEEE PROCESSOR;"
170 PRINT"  2) THE DISK PROCESSOR;"
180 INPUT"WHICH DO YOU WANT TO PEEK (1 OR 2)";D
190 PRINT"INPUT MEMORY ADDRESS"
200 PRINT"IN HEXADECIMAL:":OPEN1,8,15
210 P$=CHR$(4)+CHR$(16):R$=CHR$(224)
220 PRINT"  ▮▮▮▮                              ]"
230 INPUTZ$
240 PRINT"]";:IFLEN(Z$)<>4THENGOTO220
250 FORJ=1TO4:Y=ASC(MID$(Z$,J))
260 IFY<58THENY=Y-48
270 IFY>64THENY=Y-55
280 IFY<0ORY>16GOTO220
290 Y(J)=Y:NEXTJ:K=0:PRINT"▮▮▮▮▮▮";
300 U=Y(3)*16+Y(4):V=Y(1)*16+Y(2)
310 IFD<>2GOTO360
320 PRINT#1,C$;CHR$(U);CHR$(V);D$
330 PRINT#1,"M-W";P$;CHR$(1);R$
340 PRINT#1,"M-R";P$:GET#1,X$:IFX$=R$GOTO340
350 U=64:V=18
360 PRINT#1,"M-R";CHR$(U);CHR$(V)
370 GET#1,X$:IFX$=""THENX$=CHR$(0)
380 PRINT" ";:X=ASC(X$)/16
390 FORJ=1TO2:X%=X:X=(X-X%)*16:IFX%>9THENX%=X%+7
400 PRINTCHR$(X%+48);:NEXTJ
410 U=U+1:IFU=256THENU=0:V=V+1
420 K=K+1:IFK<LGOTO360
430 Y(0)=0:Y(4)=Y(4)+L:J=4
440 IFY(J)>15THENY(J)=Y(J)-16:J=J-1:Y(J)=Y(J)+1:GOTO440
450 PRINT:PRINT"   ";:FORJ=1TO4:Y=Y(J):IFY>9THENY=Y+7
460 PRINTCHR$(Y+48);:NEXTJ:PRINT"]":GOTO220
READY.
```

COPY/ALL will copy relative files correctly. It replaces COPY.ALL which contains a bug; it does not replace COPY ALL which is much faster if relative files are not involved.

The Basic listing mixes upper and lower case for readability; type it in without shifting. There should be lots of space for the following Machine Language program.

```
100 print"[clr] DISK.COPY.ALL       JIM BUTTERFIELD"
110 dim L2(232),L1%(232),N$(232),T%(232),T$(4)
120 data XXX,SEQ,PRG,USR,REL
130 forJ=0to4:readT$(J):nextJ
140 input"FROM UNIT  8[3 left]";F
150 gosub800
160 F$=D$
170 input"TO UNIT  9[3 left]";T
180 gosub800
190 T$=D$
200 ifF=T andF$=T$thenrun
210 gosub860
220 close1:close15:open 15,F,15:print#15,"i"+F$
230 gosub830:if E then stop:goto220
240 input"PATTERN  *[3 left]";P$
250 forJ=1to len(P$):ifmid$(P$,J,1)<>"*"thennextJ
260 P1=J-1:ifP1>0thenP$=left$(P$,P1)
270 print"HOLD DOWN 'Y' OR 'N' KEY TO SELECT"
280 print"PROGRAMS TO BE COPIED..."
290 open 1,F,3,"$"+F$
300 gosub830:ifEthenstop:goto220
```

Continued

```
310 get#1,A$:A=asc(A$+" ")
320 ifA=1orA=65thenL1=253:goto350
330 ifA=67thenL1=761:goto350
340 close1:print"?????":stop
350 forJ=1toL1:get#1,X$:nextJ
360 N=0:N1=0:R=255:Z=89
370 N$="":gosub1020:T9=Y-128:gosub1000
380 J1=1:Z$=chr$(160):forJ=1to16:get#1,X$:N$=N$+X$
390 ifX$<>Z$thenJ1=J
400 nextJ
410 gosub990:L1%=Y
420 gosub980:gosub980
430 L2=X+256*Y
440 ifT9<1orT9>4goto540
450 ifP1>0thenifP$<> left$(N$,P1)goto540
460 printN$;" ";T$(T9)
470 p=peek(151)andR
480 getZ$:ifZ$=""andP<255goto520
490 ifZ$="Y"orZ$="N"thenZ=asc(Z$):R=255:goto520
500 ifZ$=chr$(13)thenR=0:goto520
510 goto480
520 ifZ<80thenprint"[down]                     [down]":goto540
530 N=N+1:L2(N)=L2:N$(N)= left$(N$,J1):T%(N)=T9:L1%(N)=L1%
540 ifST>0goto570
550 N1=N1+1:ifN1=8thenN1=0:goto370
560 gosub1000:goto370
570 close1:close15:print" * * * * *"
580 forJ=1toN
590 L2=L2(J):T%=T%(J):ifL>L2goto640
600 print"*** output disk full"
610 input"do you have a new one";Z$
620 ifasc(Z$)<>89thenend
630 gosub860:goto590
640 open14,F,15:open15,T,15
650 printN$(J); left$("                         ",17-len(N$(J)));
660 open3,F,3,F$+":"+N$(J)+","+T$(T%)
670 input#14,E,E$,E1,E2:gosub840:ifEthenprint"** ";E$;E:goto750
680 ifT%=4thenopen4,T,4,T$+":"+N$(J)+",L,"+chr$(L1%(J)):goto700
690 open4,T,4,T$+":"+N$(J)+","+T$(T%)+",W"
700 L=L-L2:gosub830:ifEthenprint"*** ";E$;E:goto750
710 ifT%=4thensys3312:goto730
720 sys3272
730 N$(J)="":gosub830:ifEthenprint"**** ";E$;E:goto750
740 print"[down]"
750 close4:close3:close15:close14
760 nextJ
770 X=fre(0):input"another input disk ready";Z$
780 ifasc(Z$)=89goto220
790 end
800 input"drive  0[3left]";D
810 ifD*D<>Dgoto800
820 D$=chr$(D+48):return
830 input#15,E,E$,E1,E2
840 ifE=0thenE=(ST and 191):E$="*st*"
850 return
860 open15,T,15:input"want to new the output disk  n[3left]";Z$
870 ifasc(Z$)<>89goto910
880 input"disk name,id";X$,Y$
890 print#15,"n"+T$+":"+X$+","+Y$
900 gosub830:ifEthenstop:goto860
910 print#15,"I"+T$:open1,T,0,"$"+T$+":!#$%&"
920 gosub830:ifEthenstop:goto860
930 gosub980:gosub980
940 get#1,X$:ifX$<>""goto940
950 gosub980
960 L=X+Y*256:print"(";L;"blocks free )"
970 close1:close15:return
980 get#1,X$
990 get#1,X$
1000 get#1,X$
1010 X= len(X$):ifXthenX=asc(X$)
1020 get#1,X$:Y= len(X$):ifYthenY=asc(X$)
1030 return
```

Continued

Machine Language portion of program:

```
.
.:  0cc8 a2 03 20 c6 ff 20 73 0d
.:  0cd0 8e 40 02 a5 96 8d 41 02
.:  0cd8 20 cc ff a2 04 20 c9 ff
.:  0ce0 ae 40 02 20 61 0d ad 41
.:  0ce8 02 f0 da 85 96 4c cc ff
.:  0cf0 a9 00 85 5f 85 60 20 cc
.:  0cf8 ff e6 5f d0 02 e6 60 a2
.:  0d00 0e 20 c9 ff a2 50 20 61
.:  0d08 0d a2 03 20 61 0d a6 5f
.:  0d10 20 61 0d a6 60 20 61 0d
.:  0d18 a2 01 20 61 0d 20 cc ff
.:  0d20 a2 0e 20 c6 ff 20 73 0d
.:  0d28 20 cc ff e0 30 f0 01 60
.:  0d30 a2 03 20 c6 ff a0 ff c8
.:  0d38 20 73 0d a4 61 8a 99 7a
.:  0d40 02 a6 96 f0 f2 84 62 20
.:  0d48 cc ff a2 04 20 c9 ff a0
.:  0d50 ff c8 b9 7a 02 aa 20 61
.:  0d58 0d a4 61 c4 62 d0 f2 f0
.:  0d60 95 84 61 a0 10 8a 20 d2
.:  0d68 ff a5 96 29 03 f0 03 88
.:  0d70 d0 f3 60 84 61 a0 10 20
.:  0d78 e4 ff aa a5 96 29 03 f0
.:  0d80 03 88 d0 f3 60 aa aa aa

.
```

Save with Machine Language Monitor:

```
      .S "0:COPY/ALL",08,0401,0d85
```

After program has been saved, it may be loaded and saved normally with Basic commands.

PRINT ASCII FILES

```
 100 PRINT"█PRINT ASCII FILES     JIM BUTTERFIELD"
 110 INPUT "DISK FILE NAME";F$
 120 OPEN8,8,8,F$
 125 INPUT"SCREEN OR PRINTER";D$
 126 D=ASC(D$):IFD=80GOTO130
 127 IFD<>83GOTO125
 128 POKE59468,14:PRINT"█";:OPEN4,3:POKE136,0:GOTO140
 130 OPEN4,4:POKE136,17:PRINT#4,"LISTING, FILE:   ";F$:PRINT#4,"█";
 140 SYS1536
 150 PRINT#4
 160 CLOSE4:CLOSE8
 READY.
```

THE MACHINE CODE PART

```
  ?
  .
  .:   0600 A2 08 20 C6 FF 20 E4 FF
  .:   0608 A6 96 08 AA 20 CC FF 8A
  .:   0610 29 7F C9 7F F0 3A C9 1F
  .:   0618 B0 06 C9 0D F0 14 D0 1A
  .:   0620 C9 61 90 04 29 5F D0 0A
  .:   0628 C9 41 90 06 C9 60 B0 02
  .:   0630 09 80 A2 04 20 C9 FF 20
  .:   0638 D2 FF C9 0D D0 0F A5 9E
  .:   0640 29 01 D0 FA 85 9E A5 88
  .:   0648 F0 03 20 D2 FF 20 CC FF
  .:   0650 20 E1 FF 28 F0 AA 4C CC
  .:   0658 FF 46 80 0B F3 7F 00 00
  .:   0660 44 80 01 F0 7F 00 00 44
  .:   0668 00 87 26 00 00 00 AA AA
  .:   0670 AA AA AA AA AA AA AA AA
  .
  .?
```

Continued from Page 19

DISK UNIT SPECIFICATIONS

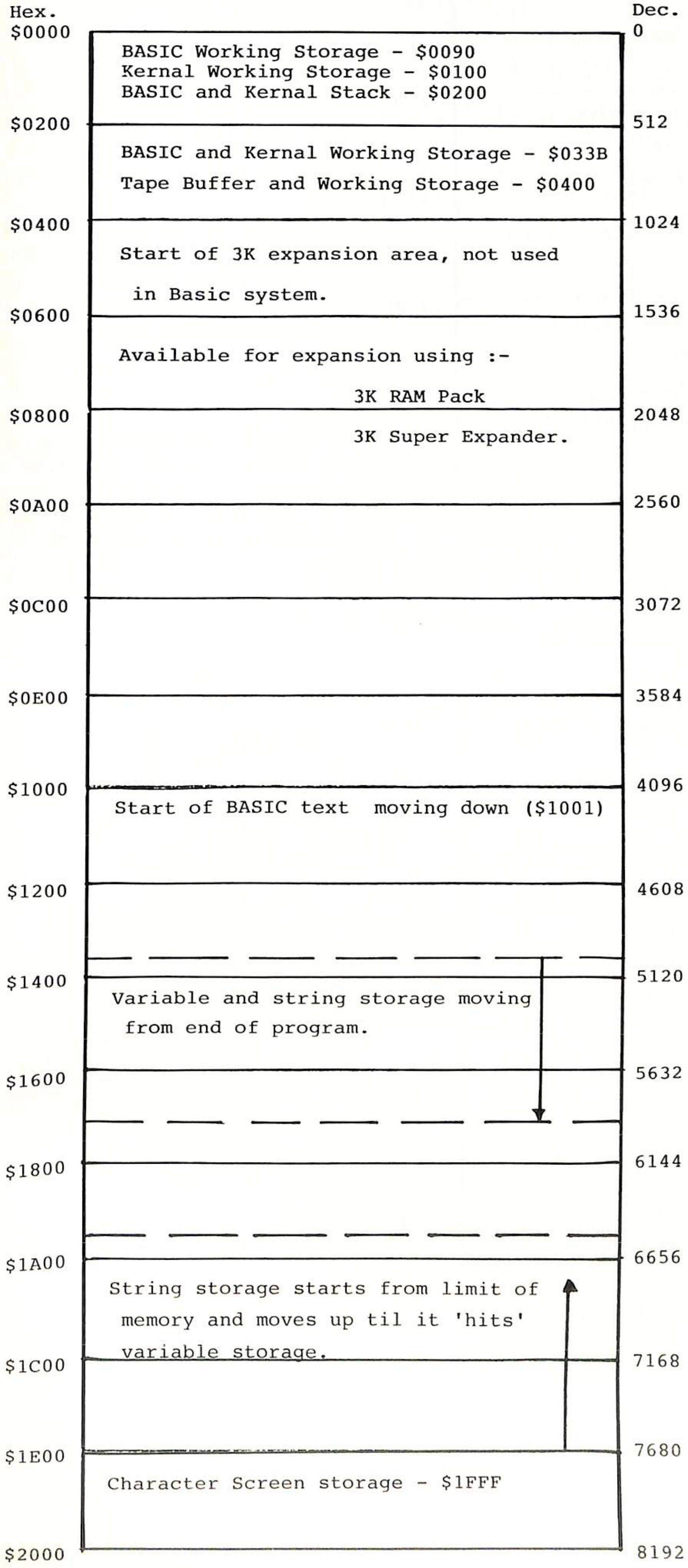
MODEL	**D9060**	**D9090**	**8250**	**8050**	**4040**	**2031**	**1540**
DRIVES	1	1	2	2	2	1	1
HEADS/DRIVE	4	6	2	1	1	1	1
INTERFACE	IEEE	IEEE	IEEE	IEEE	IEEE	IEEE	SERIAL
MEDIA*	Fixed	Fixed	DS,SD	SS,SD	SS,SD	SS,SD	SS,SD
STORAGE CAPACITY							
Unformatted	6.38Mb	9.57Mb	–	–	–	–	–
Formatted	4.97Mb	7.46Mb	2.1Mb	1.06Mb	348Kb	174Kb	174Kb
Sequential File	full disk capacity		2.1Mb	1.06Mb	174Kb	174Kb	167Kb
Relative File	full diskcapacity		1.04Mb	182Kb	171Kb	171Kb	168Kb
DISK FORMATS							
Cylinders(Tracks)	153	153	77	77	35	35	35
Sectors/Cylinder	128	192	–	–	–	–	–
Sectors per track	32	32	23-29	23-29	17-21	17-21	17-21
Bytes per sector	256	256	256	256	256	256	256
Blocks Free	19442	29162	4166	2052	664	664	664
TRANSER RATE							
Internal	5Mb/s	5Mb/s	40Kb/s	40Kb/s	40Kb/s	40Kb/s	–
IEEE-488 Bus	1.2Kb/s	1.2Kb/s	1.2Kb/s	1.2Kb/s	1.2Kb/s	1.2Kb/s	–
ACCESS TIMES (milli-seconds)							
Track-to-track	3	3	3	30	30	30	–
Average	153	153					
Head settling time	15	15					
Average Latency	8.34	8.34	100	100	100	100	–
RPM	3600	3600	300	300	300	300	–

File Interface (ALL MODELS): Shared memory – (2114) 4Kb RAM

PHYSICAL DIMENSIONS							
Height (mm)	140	140	181	181	181	144	97
Width (mm)	210	210	381	381	381	204	200
Depth (mm)	385	385	394	394	394	370	374
Weight (Kgs.)	9.5	9.5	12.6	12.6	12.6	8.0	4.5
ELECTRICAL (ALL MODELS)							
Voltage (VAC)	240	240	240	240	240	240	240
Frequency (Hz)	50	50	50	50	50	50	50
Power (watts)	200	200	70	60	60	50	25

SS Single Sided Floppy Diskette SD Single Density Floppy Diskette
DS Double sided Floppy Diskette DD Double Density Floppy Diskette

VIC MEMORY MAPS


Hex.
Dec.
$0000
0
BASIC Working Storage - $0090
Kernal Working Storage - $0100
BASIC and Kernal Stack - $0200
$0200
512
BASIC and Kernal Working Storage - $033B
Tape Buffer and Working Storage - $0400
$0400
1024
Start of 3K expansion area, not used in Basic system.
$0600
1536
Available for expansion using :-
3K RAM Pack
$0800
2048
3K Super Expander.
$0A00
2560
$0C00
3072
$0E00
3584
$1000
4096
Start of BASIC text moving down ($1001)
$1200
4608
$1400
5120
Variable and string storage moving from end of program.
$1600
5632
$1800
6144
$1A00
6656
String storage starts from limit of memory and moves up til it 'hits' variable storage.
$1C00
7168
$1E00
7680
Character Screen storage - $1FFF
$2000
8192

Lower memory map for Vic 20.

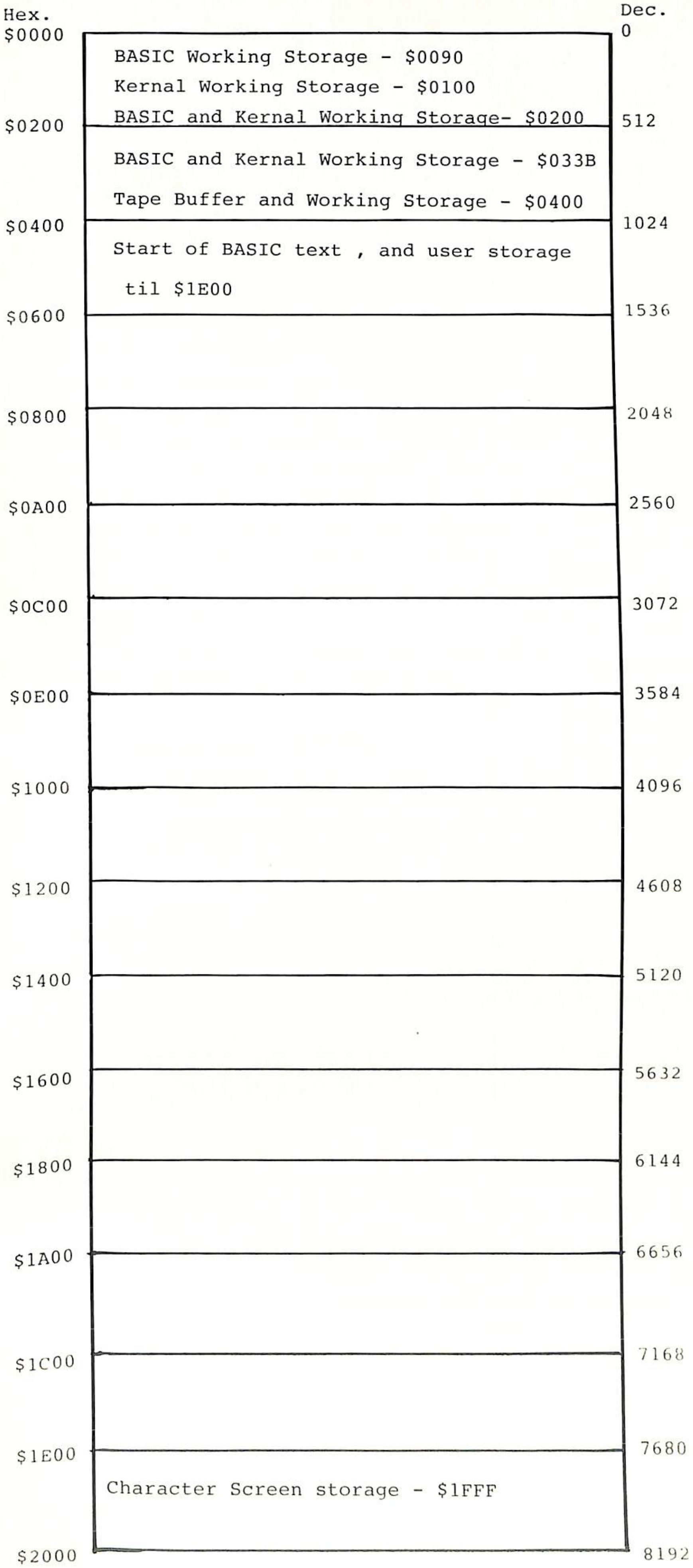
Hex.			Dec.
$2000	Block #1 includes memory from $2000 to $3FFF,		8192
$3000	RAM / ROM expansion for use with :- 8K RAM Pack, 16K RAM Pack		12288
$4000	Block #2 includes memory from $4000 to $5FFF,		16384
$5000	RAM / ROM expansion for use with :- 8K RAM Pack, 16K RAM Pack		20480
$6000	Block #3 includes memory from $6000 to $8FFF,	$6000 to $6FFF used by VICMON cartridge.	24576
$7000	RAM / ROM expan. with :- 8K RAM Pack	$7000 to $7FFF used by Programmers Aid cartridge.	28672
$8000	Reserved for Character Generator in ROM (4K).		32768
$9000	$9000 to $93FF - I/O Block & VIC Chip $9400 + 512 bytes or $9600 + 512 bytes - Colour Nibbles for screen colour.		36864
$A000	Block #4 includes memory from $A000 to $C000,		40960
$B000	RAM / ROM expansion for use with :- 8K RAM Pack Games cartridge.		45056
$C000	BASIC interpreter V2(ROM)		49152
$D000			53248
$E000	Kernal and Operating System(ROM)		57344
$F000			61440
$FFFF			65535

Upper memory map of Vic 20 with no expansion RAM in either Block #1 or at $0400.

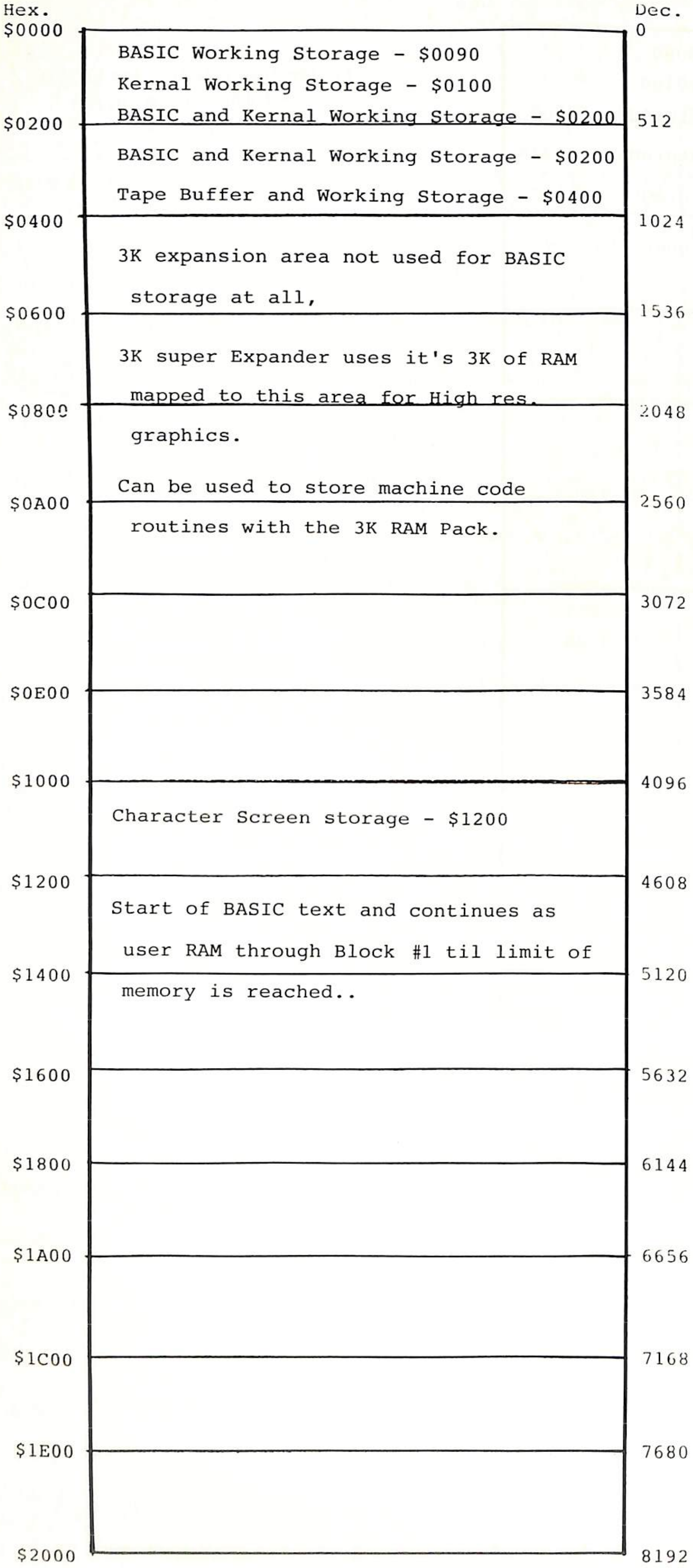
Note: 16K RAM Pack is mapped into Blocks #1 and #2.

Note: Colour Nibbles move from $9600 to $6400 when RAM is used in Block #1.

Note: $A004 to $A008 used for auto-power-up sequence.


Hex.
Dec.
$0000
0
BASIC Working Storage - $0090
Kernal Working Storage - $0100
BASIC and Kernal Working Storage- $0200
$0200
512
BASIC and Kernal Working Storage - $033B
Tape Buffer and Working Storage - $0400
$0400
1024
Start of BASIC text , and user storage til $1E00
$0600
1536
$0800
2048
$0A00
2560
$0C00
3072
$0E00
3584
$1000
4096
$1200
4608
$1400
5120
$1600
5632
$1800
6144
$1A00
6656
$1C00
7168
$1E00
7680
Character Screen storage - $1FFF
$2000
8192

Lower memory map for Vic 20 with 3k RAM Pack or Super Expander only.


Hex.
Dec.
$0000
0
BASIC Working Storage - $0090
Kernal Working Storage - $0100
BASIC and Kernal Working Storage - $0200
$0200
512
BASIC and Kernal Working Storage - $0200
Tape Buffer and Working Storage - $0400
$0400
1024
3K expansion area not used for BASIC storage at all,
$0600
1536
3K super Expander uses it's 3K of RAM mapped to this area for High res. graphics.
$0800
2048
Can be used to store machine code routines with the 3K RAM Pack.
$0A00
2560
$0C00
3072
$0E00
3584
$1000
4096
Character Screen storage - $1200
$1200
4608
Start of BASIC text and continues as user RAM through Block #1 til limit of memory is reached..
$1400
5120
$1600
5632
$1800
6144
$1A00
6656
$1C00
7168
$1E00
7680
$2000
8192

Lower memory map of Vic 20 with RAM expansion in Block #1.

Note: With this configuration the Colour Nibbles move from $9600 to $9400.

MAKE THE MOST OF YOUR COMMODORE

HIGH-RESOLUTION GRAPHICS

Now you can give your PET/CBM a High-Resolution Graphics capability with the MTU Graphics Hardware and Software Package. The Hardware is easily installed and the new graphics board provides five EXTRA ROM Sockets and 8k RAM MEMORY EXPANSION which can be used for program or data storage when graphics are not required. A powerful graphics Software Package is included and contains many extra BASIC commands for drawing lines defining shapes etc. The Graphics Hardware does not affect normal operation of the Commodore.

Available on all PET/CBM - BASIC 1, 2, 3, or 4.

RS232 TEST SET

The RS232 TESTSET is a small hand held device that connects inline with the interface cable, the terminal or the modem and monitors the line signals. The TESTSET passes all 25 lines through and so can be left connected without affecting communications. It is completely portable as no batteries are required since power us derived from the interface signals. Each indicator circuit is current limited to meet the requirements of the RS232 Interface Standard. Also the voltage range for activating the bright LED display corresponds with this standard and thereby reduces troubleshooting to a "GO-NOGO" problem instead of trying to measure active signals to determine voltage levels. One 2-pin and one 3-pin jumper are included.

PRINTOUT

Don't forget your PRINTOUT Magazine Subscription - for Commodore PET/CBM Lovers 12 issues p.a. from Jan. '82 incl. postage - $50.00 p.a.

VIC COMPUTING

For all VIC 20 owners, this sister to PRINTOUT is a must. 6 issues p.a. incl. postage from Jan. '82 - $25.00 p.a.

PROGRAMMERS TOOLKIT ROMS

These ROMs plug into spare sockets in your PET/CBMs and give the user additional commands such as TRACE, single STEP, FIND, RE-NUMBER, AUTO line numbering,DUMP variable contents, APPEND, and DELETE multiple lines. Also available are the DISK-O-PRO Tool-kits which gives all the extra DOS 2.0 commands to DOS 1.0 users as well as commands like PRINT USING, SCROLL and disk program MERGE -25 commands in all. The COMMAND-O provides DISK-O-PRO commands for BASIC 4.0 users.

The Programmers Toolkit for BASIC 1.0/2.0/3.0/4.0 users.
DISK-O-PRO Toolkit for BASIC 2.0/3.0 users.
COMMAND-O Toolkit for BASIC 4.0 users.

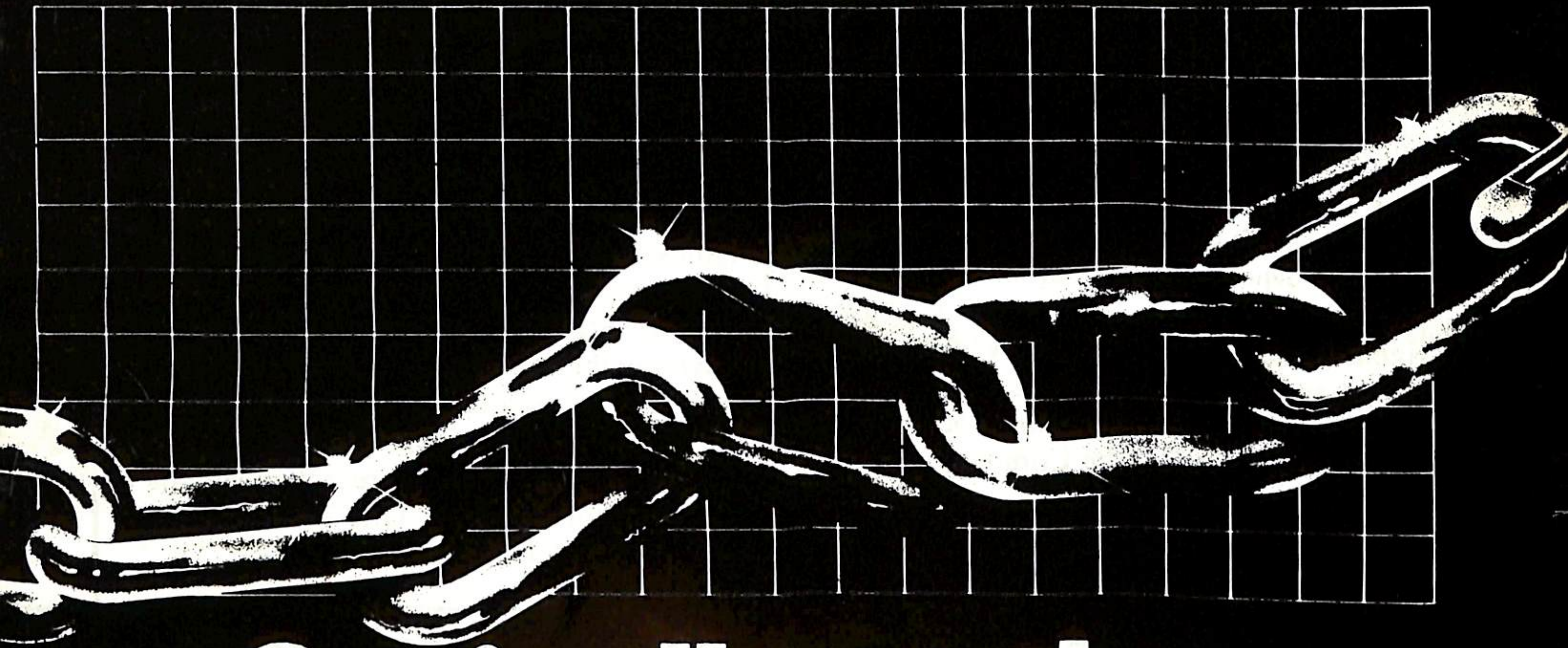
Tel: (03) 614 1433
614-1551
Telex: AA 30333.

MICROCOMP
MICROCOMPUTER SYSTEMS DESIGNERS

B.S. MICROCOMP
PTY. LIMITED,
4th & 3rd Floors,
561 Bourke Street,
Melbourne, 3000.

The Link

THAT JOINS ALL OFFICE FUNCTIONS



Get it all together with the Silicon Office.

STORING AND RETRIEVING INFORMATION – CREATING, EDITING AND PRINTING OF TEXT – MATHEMATICAL CALCULATION – COMMUNICATING INFORMATION LONG DISTANCE.

Silicon Office is the first database management System for Commodore CBM Microcomputers whereby up to six files may be open and accessed simultaneously during a run. It is also the first system which permits intercommunication with fellow machines and user. The Silicon Office turns the CBM 8032 into a secretarial work station capable of emulating any application package the user cares to think of.

Now one program which is continuously and completely resident in the memory of the CBM is capable of performing all functions required to run a small business or office. This can mean anything from Accounting and Stock Control to Word Processing, Statistical analysis, mailing lists and information filing – all at once, if necessary. Combine filing cabinets, ledgers, typewriter and calculator in your office into one efficient unit.

The Silicon Office package comprises of three integrated elements: a sophisticated word processor, a flexible database management system and an option for inter computer communications – all in one memory resident program.

$8490

THIS BUSINESS PACKAGE IS NOW AVAILABLE AND WILL COMPRISE:

- •8023 DOT MATRIX/PSEUDO PRINTER •COMMODORE CBM 8032 COMPUTER
- •8050 DISC DRIVE UNIT •64K ADD-ON MEMORY BOARD (TOTAL 96K RAM)
- •THE SILICON OFFICE PROGRAM MASTER DISC •TWO SECTIONAL A4 MANUALS

This package is available from:

Compute® CBM SYSTEMS

5 PRESIDENT AVE., CARINGBAH ☎ 525 5022